Information and Computation **163**, 60–102 (2000)
doi:10.1006/inco.2000.2891, available online at http://www.idealibrary.com on IDEAL®

Semantics of Temporal Classes[1]

Department of Computer Science, Wichita State University, Wichita, Kansas 67260-0083
E-mail: alagic@cs.twsu.edu

Published online November 16, 2000

A model theory of a typed, declarative, temporal object-oriented language system is presented. The declarative nature of the language makes it very different from the dominating procedural, strongly typed object-oriented programming languages. In this declarative system, methods are specified in a high-level, temporal constraint language. Two fundamental properties of these constraints are that they have an execution model and algebraic semantics. The model theory is based on temporal order-sorted algebras with predicates. A variety of orderings are explored in order to represent various types of inheritance, as well as the subtyping discipline. Temporal classes are viewed as temporal theories and some inheritance relationships as morphisms of temporal theories. A model of a temporal class is a temporal order-sorted structure with predicates which satisfies a set of temporal constraints specified in that class. Morphisms of those models are naturally required to preserve type coercions. A distinguished model of a temporal theory is constructed as a colimit of a suitably defined functor. This colimit construction reflects the temporal nature of the paradigm and generalizes the classical initial algebra semantics. In contradistinction to major difficulties in developing a model theory for full-fledged, typed procedural object-oriented languages, this paper shows that such a task becomes possible for a suitably defined declarative object-oriented language. This, in particular, leads to model-theoretic results on the preservation of the behavioral properties in the inheritance hierarchies.

1. INTRODUCTION

We present an algebraic model theory of a very non-traditional object-oriented language system. The language is statically typed, but unlike other typed object-oriented languages, *MyT* is a declarative language. It differs from other object-oriented languages in that it is based on a suitable temporal paradigm with

[1] This material is based upon work supported by the U.S. Army Research Office under Grant DAAH04-06-1-0192.

0890-5401/00 $35.00

well-defined (initial model) semantics, and an execution model tied to the formal semantics. As such, *MyT* complements the existing typed procedural object-oriented languages in the areas of prototyping and simulation of complex object-oriented systems. It is also intended to be a contribution to typed, object-oriented, logic-based data languages.

Research reported in this paper is significantly different from the existing results on type systems. The results of research on object-oriented type systems address largely structural properties of the object-oriented paradigm. Behavioral properties of objects remain to be specified in a conventional, procedural manner. In addition to type invariants, *MyT* allows behavioral invariants specified as temporal logic-based constraints. Such a generalized paradigm is obviously semantically much richer than the paradigms of type systems.

A fundamental requirement is that temporal constraints are not only properly typed, but also effectively executable. Their role is thus to be a step toward abandoning procedural specification of methods whenever appropriate. This is a major difference between *MyT* and specification languages. *MyT* is an executable system. In fact, in addition to the logic programming architecture, the language has an implementation model based on compilation.

An experimental implementation of *MyT* has been carried out on top of a persistent extension of the Java Virtual Machine [8]. *MyT* classes are compiled into Java class files. As such, they appear to the rest of the Java programming environment as Java classes. This accomplishes a particularly important goal—integrating *MyT* into the Java environment as a declarative, logic-based language. The issues of an optimizer and access support for persistent objects are discussed in [6].

Just like data languages, *MyT* covers a variety of situations which can be expressed in a declarative and abstract manner, free from data representation and procedural details. This allows exploratory design of complex, typed object-oriented systems. Structural prototyping is made possible by the type system, and behavioral investigations are supported by the temporal constraints. Indeed, one of the attractions of this language system is that it makes possible exploration of the behavioral properties of objects as specified in their classes for various future event patterns. This is performed by temporal queries. These query facilities [6, 8] are a fundamental distinction in comparison with procedural object-oriented languages.

Although *MyT* does not and cannot compete in its generality and efficiency with full-fledged procedural, imperative object-oriented languages, it accomplishes goals hard to attain by imperative object-oriented languages. These goals are: a statically typed design, a type system which has a semantic model, a logic basis with initial model semantics, and an execution model tied to the language semantics. Computations in *MyT* are not only type safe, but semantics of computations are precisely defined.

The focus of this paper is the underlying model theory. There have been major difficulties in developing a model theory for full-fledged, typed procedural object-oriented languages. In fact, such a model theory does not really exist at present. A contribution of this paper is to demonstrate that the task of developing an object-oriented model theory becomes possible for a suitably defined declarative object-oriented language.

The paper is organized as follows. In Section 2 we introduce the basic features of the language by examples of classes equipped with temporal constraints. In fact, Section 2 presents a typical, space-oriented application of *MyT*.

Section 3 introduces the basics of the model theory for *MyT*. The theory is based on temporal order-sorted structures with predicates. A variety of orderings are introduced. These orderings correspond to different types of inheritance. The subtyping is also naturally covered. We prove that the category of temporal-order sorted structures is equipped with the initial model, which is free over a suitably defined category of type coercions.

The temporal constraint system is the topic of Section 4. In this section we specify different forms of the allowed object-oriented temporal constraints, along with the associated deduction rules.

A model of a temporal class is a temporal order-sorted structure with predicates which satisfies the set of temporal constraints specified in that class. Section 5 defines the category of such models and constructs a distinguished model as a colimit of a suitable functor. This colimit construction reflects precisely the temporal nature of the paradigm and generalizes the classical initial model semantics approach.

In Section 6 we view temporal classes as temporal theories. This is the view "types as theories" [21] as it applies to temporal classes. In comparison with [21] this view includes a temporal as well as the object-oriented generalization. This section considers various specific forms of inheritance. Some inheritance relationships are proved to correspond to morphisms of temporal theories. A variety of standard categorical constructions are proved to be applicable in developing the presented model-theoretic results.

A problem which is very difficult to deal with in the procedural object-oriented paradigm is preservation of behavior in inheritance hierarchies. In fact, in most full-fledged object-oriented programming languages, there is no guarantee that redefinition of an inherited method preserves the semantics of the original method in the superclass. As long as the signatures of the two methods satisfy the rules of the type system, any kind of semantic change is possible. The paradigm underlying *MyT* is different in that it makes it possible to enforce only those semantic changes that are compatible with the original semantics. These issues are elaborated in Section 7.

Section 8 is a comparison of research reported in this paper with some closely related or otherwise relevant research. Section 9 is a brief summary of the most important conclusions of this paper.

2. TEMPORAL CLASSES

Methods in *MyT* are classified as observers of the object state, mutators of the object state, or constructors of new objects. The effects of mutators and constructors on observers are defined by temporal constraints. Object-oriented programming in *MyT* reduces to writing object-oriented, temporal constraints.

2.1. Methods and constraints

The main construct of *MyT* is the class specification block as a unit of encapsulation, information hiding, inheritance and polymorphism. A specification block defines an object type (class) and includes the following components:

- A collection of *observers* of the underlying object state. Observers are predicates whose result type is thus omitted from class specifications. The role of observers is information hiding, as the actual representation of the object state is completely hidden from the users. The users can inspect only the externally visible properties of the hidden object state by invoking observer predicates or more general temporal queries.
- A collection of *constructors*. Constructors are functions, and as such they are used to specify a variety of operators. The result of a constructor application is an object with a new identity.
- A collection of *mutators* that affect the underlying object state, while preserving the object identity. Mutators are thus state update events, and are consequently modeled in an event-oriented style, using predicates.
- *Constraints* expressed in temporal Horn clause logic. Temporal constraints specify the effect of update (mutator) events (simple or composite) on the underlying object state, as visible via observer predicates. Likewise, constraints specify the observable properties of objects created by constructor messages.

The constraint language contains three temporal operators. The operator *always* is denoted as $\Box$. If ζ is a constraint, then $\Box\zeta$ is true iff ζ evaluates to true in all object states. The operator *next time* is denoted as $\bigcirc$. The constraint $\bigcirc\zeta$ is true in the current state iff the constraint ζ is true in the next object state. The operator *some time* is denoted as $\Diamond$. $\Diamond\zeta$ is true iff there exists a state, either the current one or a future one, in which ζ evaluates to true.

The constraints are expressed by temporal Horn clauses. Standard Horn clauses have the form $A \leftarrow B_1, B_2, ..., B_n$, where A is the head, $B_1, B_2, ..., B_n$ is the body of the clause, $A, B_1, B_2, ..., B_n$ are atomic predicates, $\leftarrow$ denotes implication and comma denotes conjunction. In addition, *MyT* allows the three temporal operators to appear in temporal Horn clauses. There are restrictions on the usage of the temporal operators in the constraint language. These restrictions are explained in Section 4. The restrictions limit the expressive power of the language, but at the same time they guarantee the existence of an execution model of the language [8] and its formal semantics.

Our first illustration of a class with temporal constraints is *MovingObject*. The observers of a moving object naturally include its coordinates. Three additional observers define the magnitude of the speed vector, and two angles, heading and pitch. The associated temporal constraints relate the position, speed, heading, and pitch of a moving object to its position in the next time instant. There is one constructor, *clone*, which creates a new moving object with the position specified by the arguments of this message and, with the same speed, heading and pitch as the receiver object of the clone message.

EXAMPLE 1. (Temporal class)

Class MovingObject
Observers
x(Number), y(Number), z(Number),
speed(Number),
heading(Number), pitch(Number)
Constructors
clone(Number,Number,Number): MyClass
Constraints
ForAll self: MyClass, X, Y, Z: Number, S: Number, D, G: Number:
(**transition constraints**)
□(○self.x(X.add(S.times (D.cos()).times(G.cos()))
← self.x(X), self.speed(S), self.heading(G),
self.pitch(D)),
□(○self.y(Y.add(S.times (D.cos()).times(G.sin()))
← self.y(Y), self.speed(S), self.heading(G),
self.pitch(D)),
□(○self.z(Z.add(S.times (D.sin()))
← self.z(Z), self.speed(S), self.pitch(D)),
(**constructor constraints**)
□self.clone(X, Y, Z).x(X) ←,
□self.clone(X, Y, Z).y(Y) ←,
□self.clone(X, Y, Z).z(Z) ←,
□(self.clone(X, Y, Z).heading(G) ← self.heading(G)),
□(self.clone(X, Y, Z).speed(S) ← self.speed(S)),
□(self.clone(X, Y, Z).pitch(D) ← self.pitch(D))
End MovingObject.

The type of objects, instances of a class, is denoted by *MyClass*, in the specification of that class. *self* denotes the receiver object, i.e., the object executing the message. The type of *self* is, of course, *MyClass*. Although redundant, this fact is made explicit in sample classes given in the paper.

Note that no mutators are specified in the class *MovingObject*. Also, the constraints are given under the assumption of a sufficiently small interval between two successive time instances. In sample temporal classes in this paper initial constraints are omitted for reasons of simplicity. These constraints specify the observable properties of newly created objects. The form of the initial constraints is specified in Section 4.1.

2.2. Inheritance

A class *Satellite* is derived below by inheritance from the class *MovingObject*. Satellites are moving objects with a fixed elliptic trajectory. The axes of the trajectory are rigid (time independent) properties of a satellite object. The same applies to its speed. Rigidity of these observers is expressed using the temporal operator ◇ in the body and the temporal operator □ in the head. Since the trajectory and the

speed are rigid, there are no mutators in the class *Satellite* that would affect these observers. For the sake of simplification, we have chosen the coordinate system in such a way that its center coincides with the center of the ellipsis, so that the third coordinate of a satellite is 0 at all times.

EXAMPLE 2. (Inheritance)

Class Satellite
Inherits MovingObject
Observers
axisX(Number), axisY(Number)
Constraints
ForAll self: MyClass, X, Y, Ax, Ay: Number, S: Number:
(**class invariants**)
□self.z(0) ←,
□self.pitch(0) ←,
(**history properties**)
□self.axisX(Ax) ← ◇self.axisX(Ax),
□self.axisY(Ay) ← ◇self.axisY(Ay),
□self.speed(S) ← ◇self.speed(S),
(**class invariant**)
□(self.heading(X.times(axisY.sqr()).div(Y.times(axisX.sqr()).minus().arctan())
← self.x(X), self.y(Y), self.axisX(Ax), self.axisY(Ay))
End Satellite.

A class specification consists of signatures for functions and predicates encapsulated with temporal constraints. As explained formally in Section 6, this makes a temporal class a theory. For the model theory presented in this paper the following two specific situations of using inheritance to derive one temporal class from another have particular importance:

- *MyClass* appears in the superclass, and thus signatures for functions and predicates are redefined in the subclass. Temporal constraints are inherited and suitably extended to reflect the change of interpretation of *MyClass* in the subclass. Signatures for additional functions and predicates, together with the associated constraints, may be introduced in the subclass.

- *MyClass* is not used. All the signatures and the associated constraints are inherited in the subclass in the form in which they appear in the superclass. Signatures for additional functions and predicates, together with the associated constraints, are introduced in the subclass.

The second case above is a particular case of subtyping. However, it should come as no surprise that there are difficulties in applying parts of the material on temporal theories in Section 6 to subtyping in general. The same holds for those situations in which the semantics of the inherited methods is completely changed, as opposed to being extended in a monotonic fashion.

2.3. Behavioral Patterns

The temporal object-oriented paradigm of *MyT* allows declarative specification of different behavioral patterns that objects of different classes conform to. As an illustration, we specify a very different pattern of behavior as exhibited by missile objects. The behavior of a missile object is determined by the behavior of another moving object, its target. The level of control of movement of a missile is reduced to setting its target. Once the target is determined invoking the mutator *setTarget*, the movement of a missile is entirely determined by the movement of the target.

EXAMPLE 3. (Mutators)

Class Missile
Inherits MovingObject
Observers
target(MovingObject)
Mutators
setTarget(MovingObject)
Constraints
ForAll self: MyClass, T: MovingObject, X1, X2, Y1, Y2, Z1, Z2: Number:
(**transition constraint**)
□(○self.target(T) ← self.setTarget(T)),
(**class invariants**)
□(self.heading(Y2.minus(Y1).div(X2.minus(X1)).arctan())
← self.target(T), T.x(X2), T.y(Y2), self.x(X1), self.y(Y1)),
□(self.pitch(Z2.minus(Z1).div(X2.minus(X1).sqr().plus(Y2.minus(Y1).sqr()).sqrt())
← self.target(T), T.x(X2), T.y(Y2), T.z(Z2), self.x(X1), self.y(Y1),
self.z(Z1))
End Missile.

In order for a constraint section to be complete, the effect of each mutator on each observer in the next object state should be specified. Likewise, completeness requires specification of all the observable properties of objects created by the class constructors. These conditions are checked at compile-time. This way all the observable effects of all mutator and constructor messages on the underlying hidden object state are completely specified.

Explicit and separate specification of mutators carries an obvious underlying assumption: if no mutators are executed, the underlying object state remains the same. The observers are thus not affected. Otherwise, frame axioms of this type require temporal clauses with negation in the body.

The third behavioral pattern derived from the generic moving object pattern is that of a space shuttle. A shuttle (we really mean the orbiter part) is an object whose movements can be completely controlled by mutators *changeSpeed* (subject to an additional constraint which defines the maximum speed as a rigid property), *changeHeading*, and *changePitch*.

EXAMPLE 4. (Mutators)

Class Shuttle
Inherits MovingObject
Observers
maxSpeed(Number),
airborne()
Mutators
changeSpeed(Number),
changeHeading(Number),
changePitch(Number)
Constraints
ForAll self: MyClass, X, Y, Z: Number, Max, S, S': Number, D, D', G, G': Number:
(**class invariant**)
□(self.airborne() ← self.z(Z), Z.greaterThan(0)),
(**history property**)
□self.maxSpeed(S) ← ◇self.maxSpeed(S);
(**transition constraints**)
□(○self.speed(S.add(S'))
← self.speed(S), self.maxSpeed(Max), S.add(S').lessThanEq(Max),
self.changeSpeed(S')),
□(○self.heading(G.add(G')) ← self.heading(G), self.changeHeading(G')),
□(○self.pitch(D.add(D')) ← self.pitch(D), self.changePitch(D'))
End Shuttle.

2.4. Constrained Matching

When *MyClass* appears in the argument position of a method, it gets covariantly redefined when the method is inherited. This causes well known problems in the type system [14]. A technique called constrained matching that is provably type safe [7] is used in *MyT* in order to avoid well-known anomalies. This technique relies on the temporal constraint system in order to avoid problems that cannot be handled by the type system. Constrained matching is the topic of a separate paper [7]. It will be illustrated here by a classical example given below:

EXAMPLE 5. (Covariance)

Class TwoDObject
Observers
x(Number), y(Number)
Mutators
move(Number)
Constructors
mid(MyClass): MyClass
Constraints
ForAll self, P2: MyClass, X1, X2, Y1, Y2: Number:
□self.x(X1) ← ◇self.x(X1),

□(○self.y(Y1.add(Y2)) ← self.y(Y1), self.move(Y2)),
□(self.mid(P2).x(X1.add(X2).div(2)) ← self.x(X1), P2.x(X2)),
□(self.mid(P2).y(Y1.add(Y2).div(2)) ← self.y(Y1), P2.y(Y2))
End TwoDObject.

A class *ThreeDObject* is now derived by inheritance from the class *TwoDObject*:

EXAMPLE 6. (Covariance)

Class ThreeDObject
Inherits TwoDObject
Observers
z(Number)
Constraints
ForAll self, P3: MyClass, Z1, Z2: Number:
□self.z(Z1) ← ◇self.z(Z1),
□(self.mid(P3).z(Z1.add(Z2).div(2)) ← self.z(Z1), P3.z(Z2))
End ThreeDObject.

ThreeDObject does not define a subtype of *TwoDObject*. If we allow substitution of an object of the class *ThreeDObject* in place of an object of the class *TwoDObject*, in a system with single dispatch and dynamic binding, type safety cannot be guaranteed by static type checking. With *P*2: *TwoDObject*, the expression *P*2.*mid*(*P*2) obviously satisfies the static type check. But at run time the receiver *P*2 may in fact be an object of a class derived by inheritance from *TwoDObject*, such as *ThreeDObject*. Single dispatch will select the method based on the run-time type of the receiver. If that type is *ThreeDObject*, in a conventional system this will cause a run-time failure due to a wrong type of the argument.

Consider now a class *ThreeDObject* derived by constrained matching from the class *TwoDObject*:

EXAMPLE 7. (Constrained matching)

Class ThreeDObject
Inherits TwoDObject
Observers
z(Number)
Constructors
mid(TwoDObject): MyClass
Constraints
ForAll self, P3: MyClass, P2: TwoDObject, Z1, Z2: Number:
□self.z(Z1) ← ◇self.z(Z1),
□(self.mid(P2).z(Z1.div(2)) ← self.z(Z1)),
□(self.mid(P3).z(Z1.add(Z2).div(2)) ← self.z(Z1), P3.z(Z2))
End ThreeDObject.

Subtyping requires that the argument type of the method *mid* in the class *ThreeDObject* is a subtype of *TwoDObject*. This effect is achieved in our formal system by an additional (overloaded) method signature *mid*(*TwoDObject*): *MyClass*

and a constraint $\Box(self.mid(P2).z(Z1.div(2)) \leftarrow self.z(Z1))$ for $P2$: *TwoDObject*. This way the method *mid* in the class *ThreeDObject* is defined both for the arguments of types *ThreeDObject* and *TwoDObject*. A formal presentation of the type system for *MyT* and the constrained matching technique is the topic of a separate paper [7]. This paper concentrates on the model theory.

The usual bottom-up search for locating the relevant method definition starts with the run-time type of the receiver object. In our system it is augmented with inspection of the run-time types of the arguments (multiple dispatch). This way the correct method definition is always located. The operational model is discussed further in Section 7.

3. TEMPORAL ORDER-SORTED STRUCTURES

The model theory developed in this paper is based on temporal order-sorted structures. These structures are objects of a category equipped with the initial object. A distinctive property of this object is that all other models in the category are its homomorphic images. We provide a careful characterization of this result which requires the category of type coercions. The latter captures permissible substitutions. A variety of such substitutions corresponding to various types of inheritance and subtyping are considered.

The model theory is based on the following rules of correspondence:

Rules (Rules of correspondence).

- Each class is assigned a distinct sort.
- The receiver of the message is viewed as the first argument of a predicate or a function as defined below.
- An n-ary observer of a class C with signature $p(C_1, C_2, ..., C_n)$ is viewed as an $(n+1)$-ary predicate with signature $p(C, C_1, C_2, ..., C_n)$.
- The same rule applies to n-ary mutators.
- An n-ary constructor of a class C with signature $f(C_1, C_2, ..., C_n) : C_m$ is viewed as an $(n+1)$-ary function with signature $f(C, C_1, C_2, ..., C_n) : C_m$.
- The inheritance relationships among classes are viewed as suitably defined partial orders of the set of sorts. Different specific situations are covered by different specific orderings.

3.1. Order-Sorted Signature

A temporal class contains, first of all, specification of the signatures for constructors, observers, and mutator methods. Formally, this part of the specification of a temporal class is captured by the notion of an order-sorted signature. Such a signature contains function signatures for constructors and predicate signatures for observers and mutators. Sorts in such an order-sorted signature correspond to all the classes referred to in a temporal class.

DEFINITION 1. An *order-sorted signature* is a tuple $(S, \leqslant, \Sigma, \Pi)$ such that:

1. S is a set whose elements are called sorts.
2. S is equipped with a partial ordering $\leqslant$ which extends to $S^* \times S$.
3. Σ is an $S^* \times S$-sorted family $\{\Sigma_{w,s} \mid w \in S^*, s \in S\}$ of function symbols. When $\sigma \in \Sigma_{w,s}$, we say that σ has *rank* $w \to s$ (or, just (w, s)), arity w and sort s. If $w = \lambda$ (the empty string), then $\Sigma_{\lambda,s}$ is the set of constants of sort s.
4. The following *monotonicity* condition is satisfied: $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ and $w \leqslant w'$ imply $s \leqslant s'$.
5. $\Pi = \{\Pi_w \mid w \in S^*\}$ is a family of *predicate symbols*.

3.2. A Variety of Orderings

Object-oriented languages specify different types of relationships among classes. The two most common are inheritance and subtyping. In some type systems (Java and C++), these two are intended to coincide. In other type systems (for example, Eiffel) the inheritance relationships do not correspond to subtyping. Neither does the relationship called matching [13]. This variety of relationships is first formally defined in this section in terms of orderings of sorts of an order-sorted signature. The properties of these orderings will be studied in subsequent sections.

DEFINITION 2. If $\sqsubseteq$, $<:$, $\preccurlyeq$, and $<=$ are partial orders on the set of sorts S, then their extensions to $S^* \times S$ are defined as follows:

1. *Covariant ordering* (Eiffel):

$$(s_1 s_2 \cdots s_n, s) \sqsubseteq (s'_1 s'_2 \cdots s'_n, s') \text{ iff } s_j \sqsubseteq s'_j \text{ for } 1 \leqslant j \leqslant n \text{ and } s \sqsubseteq s.'$$

2. *Contravariant* (*subtyping*) *ordering* (Trellis-Owl, Modula-3):

$$(s_1 s_2 \ldots s_n, s) <: (s'_1 s'_2 \ldots s'_n, s') \text{ iff } s_1 <: s'_1,\ s'_j <: s_j \text{ for } 2 \leqslant j \leqslant n \text{ and } s <: s'.$$

3. *Matching* (PolyToil [13]): If *MyClass* is viewed as a fixed sort, we obtain a very simple definition of this type of orderings:

$$(s_1 s_2 \ldots s_n, s) \preccurlyeq (s'_1 s'_2 \ldots s'_n, s') \text{ iff } (s_1 s_2 \ldots s_n, s) <: (s'_1 s'_2 \ldots s'_n, s').$$

Note that this is just a formal, syntactic definition. The semantics of *MyClass* is such that it changes its interpretation in a class derived by inheritance.

4. *Constrained matching* (*MyT*): This is a limited form of matching in which the underlying class is always denoted by *MyClass*. Explicit references to the name of the underlying class are not allowed and neither are the changes of the method signatures. The only change is caused by the change of interpretation of *MyClass*. If we view *MyClass* as a fixed sort, we obtain a very simple syntactic definition for constrained matching:

$$(s_1 s_2 \ldots s_n, s) \preccurlyeq (s'_1 s'_2 \ldots s'_n, s') \text{ iff for } j = 1, \ldots, n,\ s_j = s'_j \text{ and } s = s'.$$

Note that we use the same symbol for matching and constrained matching. The reason is that we will be concerned with constrained matching alone because of its desirable semantic properties.

5. *Conservative ordering* (C++, Java):

$$(s_1 s_2 \ldots s_n, s) <= (s'_1 s'_2 \ldots s'_n, s') \text{ iff}$$

- $s_1 <= s'_1$
- $s_j = s'_j$ for $2 \leqslant j \leqslant n$
- $s = s'$.

PROPOSITION 1. *All the extended orderings are partial orders.*

3.3. Order-Sorted Temporal Structures

The first step in developing a model theory for temporal classes is to consider suitable temporal models for order-sorted signatures with predicates. In the well-established approach [20, 21, 25], models of order-sorted signatures with predicates are equipped with a relation for each predicate. Temporal order-sorted structures generalize these models by having a sequence of relations for each predicate, one relation per time instant.

DEFINITION 3. An order-sorted temporal structure [5] $M = (M_S, \Sigma, \Pi)$ over an order-sorted signature $(S, \leqslant, \Sigma, \Pi)$ consists of:

1. A family $M_S = \{M_s \mid s \in S\}$ of sets called the carriers.
2. A family of coercion functions $M_s \to M_{s'}$ whenever $s \leqslant s'$.
3. For every $\sigma \in \Sigma_{w,s}$, a function $f_\sigma : M_w \to M_s$, where $M_w = M_{s1} \times \cdots \times M_{sn}$ when $w = s_1 s_2 \cdots s_n$, and where M_w is a one element set when $w = \lambda$.
4. The following coercion condition is required

$$\begin{array}{ccc} M_w & \xrightarrow{f_\sigma} & M_s \\ {\scriptstyle c_{w,w'}}\downarrow & & \downarrow{\scriptstyle c_{s,s'}} \\ M_{w'} & \xrightarrow{f_\sigma} & M_{s'} \end{array}$$

whenever $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ and $w \leqslant w'$.

5. For each predicate A in Π_w, a sequence of subsets (relations) $M^w_{A_i} \subseteq M_w$, $i \in N_0$.

A particular and very important case occurs when coercions are in fact embeddings. Conditions 2 and 4 generalize our earlier results reported in [5]. Condition 5 is a temporal generalization of the predicate condition as defined in [25].

3.4. Temporal Order-Sorted Morphisms

A morphism of temporal order-sorted structures is a generalization of a well-established notion for order-sorted structures [20, 21, 25]. The generalization applies to the relations as they model predicates. As opposed to one condition per predicate, a morphism of temporal order-sorted structures has a sequence of conditions, one condition per time instant.

DEFINITION 4. Let $M = (M_S, \Sigma, \Pi)$ and $M' = (M'_S, \Sigma, \Pi)$ be temporal order-sorted structures over the same Σ, Π signature. A temporal order-sorted morphism $h: M \to M'$ is a family of s-sorted functions $h = \{h_s: M_s \to M'_s \mid s \in S\}$ satisfying the following conditions:

1. *Morphism condition*: For each $\sigma \in \Sigma_{w,s}$, $a \in M_w$, $h_s(f_\sigma(a)) = f'_\sigma(h_w(a))$ holds, where h_w is a product of functions $h_{s1} \times \cdots \times h_{sn}$ when $w = s_1 s_2 \cdots s_n$, as in the diagram below:

$$\begin{array}{ccc} M_{s1} \times \cdots \times M_{sn} & \xrightarrow{f_\sigma} & M_s \\ {\scriptstyle h_{s1} \times \cdots \times h_{sn}}\downarrow & & \downarrow{\scriptstyle h_s} \\ M'_{s1} \times \cdots \times M'_{sn} & \xrightarrow{f'_\sigma} & M'_s \end{array}$$

2. *Coercion condition*

$$\begin{array}{ccc} M_s & \xrightarrow{h_s} & M'_s \\ {\scriptstyle c_{s,s'}}\downarrow & & \downarrow{\scriptstyle c_{s,s'}} \\ M_{s'} & \xrightarrow{h_{s'}} & M'_{s'} \end{array}$$

whenever $s \leqslant s'$.

3. *Temporal* (*predicate*) *condition*: For each $A \in \Pi_w$ where $w = s_1 s_2 \cdots s_n$, $h_w(a_1, a_2, ..., a_n) \in M'^w_{A_i}$ whenever $(a_1, a_2, ..., a_n) \in M^w_{A_i}$, for $i \in N_0$, as in the diagram below:

$$\begin{array}{ccc} M^w{}_{A_i} & \xrightarrow{\subseteq} & M_{s1} \times \cdots \times M_{sn} \\ {\scriptstyle h_w}\downarrow & & \downarrow{\scriptstyle h_{s1} \times \cdots \times h_{sn}} \\ M'^w_{A_i} & \xrightarrow{\subseteq} & M'_{s1} \times \cdots \times M'_{sn} \end{array}$$

Condition 2 generalizes our earlier results reported in [5]. Condition 3 is a temporal generalization of the predicate condition as defined in [25].

PROPOSITION 2. *Temporal order sorted Σ, Π structures and their Σ, Π morphisms form a category, denoted as* $\mathbf{Mod}_{\Sigma,\Pi}$.

3.5. Order-Sorted Temporal Term Algebra

Among temporal order-sorted structures for a given order-sorted signature with predicates, there exists a distinguished one. This distinguished temporal order-sorted structure has the property that any other temporal structure in the same category (i.e., for a fixed order-sorted signature) is its image under a unique morphism. This makes this distinguished temporal order-sorted structure the initial temporal order-sorted structure in its category.

Let $\mathscr{X}$ be an S-sorted set of variables disjoint from Σ such that $\mathscr{X}_s$ denotes the set of variables of sort s. $\Sigma(\mathscr{X})$ denotes an S-sorted signature obtained by enlarging the set of constants in Σ with $\mathscr{X}$ as follows:

- $\Sigma(\mathscr{X})_{\lambda, s} = \Sigma_{\lambda, s} \cup \mathscr{X}_s$;
- $\Sigma(\mathscr{X})_{w, s} = \Sigma_{w, s}$ for $w \neq \lambda$; i.e., only the sets of constants are enlarged.

DEFINITION 5. An order-sorted temporal Σ, Π-term algebra $\mathscr{T}_\Sigma(\mathscr{X})$ is the least family $\{\mathscr{T}_{\Sigma, s}(\mathscr{X}) \mid s \in S\}$ of sets of terms satisfying the following conditions:

1. The carriers are defined as follows:

- $\Sigma(\mathscr{X})_{\lambda, s} \subseteq \mathscr{T}_{\Sigma, s}(\mathscr{X})$ for $s \in S$, i.e., terms of depth zero are constants and variables.
- If $\sigma \in \Sigma_{w, s}$ and if $t_j \in \mathscr{T}_{\Sigma, sj}(\mathscr{X})$ for $j = 1, 2, ..., n$, where $w = s_1 s_2 \cdots s_n \neq \lambda$, then $\sigma(t_1 \ldots t_n) \in \mathscr{T}_{\Sigma, s}(\mathscr{X})$

2. Coercions have a very particular form:

$\mathscr{T}_{\Sigma, s}(\mathscr{X}) \subseteq \mathscr{T}_{\Sigma, s'}(\mathscr{X})$ if $s \leqslant s'$

3. Functions symbols are interpreted as follows: for $\sigma \in \Sigma_{w, s}$, $f_\sigma : \mathscr{T}_{\Sigma, w}(\mathscr{X}) \to \mathscr{T}_{\Sigma, s}(\mathscr{X})$ maps $t_1, ..., t_n$ to $\sigma(t_1 ... t_n)$, where $\mathscr{T}_{\Sigma, w}(\mathscr{X}) = \mathscr{T}_{\Sigma, s1}(\mathscr{X}) \times \cdots \times \mathscr{T}_{\Sigma, sn}(\mathscr{X})$

4. Predicates are interpreted as empty relations: $(\mathscr{T}^w_\Sigma(\mathscr{X}))_{A_i} = \varnothing$ $(i \in N_0)$ for each $A \in \Pi_w$.

3.6. Temporal order-sorted initial structure

DEFINITION 6. Let $M = (M_S, \Sigma, \Pi)$ be a temporal order sorted structure, $\mathscr{X}$ and $\mathscr{Y}$ sets of S-sorted variables.

1. A family $h : \mathscr{X} \to M$ of S-sorted functions satisfying the following coercion condition

$$\begin{array}{ccc} \mathscr{X}_s & \xrightarrow{h_s} & M_s \\ {\scriptstyle c_{s, s'}}\downarrow & & \downarrow{\scriptstyle c_{s, s'}} \\ \mathscr{X}_{s'} & \xrightarrow{h_{s'}} & M_{s'} \end{array}$$

is called an *assignment of variables.*

2. An assignment of variables of the form $h: \mathscr{Y} \to \mathscr{T}_{\Sigma}(\mathscr{X})$ is called a *substitution of variables.*

DEFINITION 7. M is a *free temporal order-sorted Σ, Π structure* over $\mathscr{X}$ if for any temporal structure $M' = (M_S', \Sigma, \Pi)$, any assignment of variables $h: \mathscr{X} \to M'$ extends to a unique temporal order-sorted morphism $h^*: M \to M'$ such that $h^*(X) = h(X)$ for $X \in \mathscr{X}$.

DEFINITION 8. An order-sorted Σ, Π temporal structure M is *initial* in the category of Σ, Π temporal order-sorted structures iff there exists a unique temporal order sorted morphism from M to any other Σ, Π temporal order sorted structure.

THEOREM 1. (Free temporal structure). *The temporal structure* $\mathscr{T}_{\Sigma} = (\mathscr{T}_{\Sigma, \Pi}(\mathscr{X}), \Sigma, \Pi)$ *is a free order-sorted temporal* (Σ, Π) *structure over* $\mathscr{X}$.

Proof 1. We must show that for any temporal order-sorted Σ, Π structure $M = (M_S, \Sigma, \Pi)$, and any assignment of variables $h: \mathscr{X} \to M$, there is a unique temporal order-sorted (Σ, Π) morphism $h^*: (\mathscr{T}_{\Sigma, \Pi}(\mathscr{X}), \Sigma, \Pi) \to (M_S, \Sigma, \Pi)$ such that $h^*(X) = h(X)$ for $X \in \mathscr{X}$. Define h^* inductively as $h^*(\sigma(t_1, t_2, ..., t_n)) = f_{\sigma}(h^*(t_1), h^*(t_2), ..., h^*(t_n))$. We thus have the following commutative diagrams:

$$\begin{array}{ccc} \mathscr{X}_s & \xrightarrow{\subseteq} & \mathscr{T}_{\Sigma, \Pi}(\mathscr{X})_s \\ {\scriptstyle =}\downarrow & & \downarrow{\scriptstyle h_s^*} \\ \mathscr{X}_s & \xrightarrow{h_s} & M_s \end{array} \qquad \begin{array}{ccc} \mathscr{T}_{\Sigma, \Pi}(\mathscr{X})_w & \xrightarrow{\sigma} & \mathscr{T}_{\Sigma, \Pi}(\mathscr{X})_s \\ {\scriptstyle h_w^*}\downarrow & & \downarrow{\scriptstyle h_s^*} \\ M_w & \xrightarrow{f_{\sigma}} & M_s \end{array}$$

1. The morphism condition is satisfied by the above definition of h^*.

2. In order to prove that the coercion condition is satisfied as well, we proceed by induction on the depth of terms. The first step is the following diagram

$$\begin{array}{ccc} \mathscr{X}_s & \xrightarrow{h_s^*} & M_s \\ {\scriptstyle c_{s, s'}}\downarrow & & \downarrow{\scriptstyle c_{s, s'}} \\ \mathscr{X}_{s'} & \xrightarrow{h_{s'}^*} & M_{s'} \end{array}$$

which is commutative since $h^*(X) = h(X)$ for $X \in \mathscr{X}$.
The inductive hypothesis is

$$\begin{array}{ccc} \mathscr{T}_{\Sigma, \Pi}(\mathscr{X})_w & \xrightarrow{h_w^*} & M_w \\ {\scriptstyle c_{w, w'}}\downarrow & & \downarrow{\scriptstyle c_{w, w'}} \\ \mathscr{T}_{\Sigma, \Pi}(\mathscr{X})_{w'} & \xrightarrow{h_{w'}^*} & M_{w'} \end{array}$$

where $w \leqslant w'$. The inductive step which must be proved is expressed by the following two diagrams:

$$\begin{array}{ccccc}
\mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_w & \xrightarrow{\sigma} & \mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_s & \xrightarrow{h_s^*} & M_s \\
\downarrow{\scriptstyle c_{w,w'}} & & \downarrow{\scriptstyle c_{s,s'}} & & \downarrow{\scriptstyle c_{s,s'}} \\
\mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_{w'} & \xrightarrow{\sigma} & \mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_{s'} & \xrightarrow{h_{s'}^*} & M_{s'}
\end{array}$$

The proof also requires the coercion condition for $M = (M_S, \Sigma, \Pi)$:

$$\begin{array}{ccc}
M_w & \xrightarrow{f_\sigma} & M_s \\
\downarrow{\scriptstyle c_{w,w'}} & & \downarrow{\scriptstyle c_{s,s'}} \\
M_{w'} & \xrightarrow{f_\sigma} & M_{s'}
\end{array}$$

Two more diagrams are required in order to complete the proof, both implied by the definition of h^*.

$$\begin{array}{ccc}
\mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_w & \xrightarrow{\sigma} & \mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_s \\
\downarrow{\scriptstyle h_w^*} & & \downarrow{\scriptstyle h_s^*} \\
M_w & \xrightarrow{f_\sigma} & M_s
\end{array}
\qquad
\begin{array}{ccc}
\mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_{w'} & \xrightarrow{\sigma} & \mathscr{T}_{\Sigma,\Pi}(\mathscr{X})_{s'} \\
\downarrow{\scriptstyle h_{w'}^*} & & \downarrow{\scriptstyle h_{s'}} \\
M_{w'} & \xrightarrow{f_\sigma} & M_{s'}
\end{array}$$

3. The predicate condition is trivially satisfied since $(\mathscr{T}_\Sigma^w(\mathscr{X}))_{A_i} = \varnothing$ $(i \in N_0)$ for each $A \in \Pi_w$.

COROLLARY 1 (Initial temporal structure). *The temporal structure* $(\mathscr{T}_{\Sigma,\Pi}(\varnothing), \Sigma, \Pi)$ *is the initial temporal* Σ, Π *structure.*

Proof 2. Immediate from the previous theorem setting $\mathscr{X} = \varnothing$.

The characterization of the distinctive property of the initial model must take into account permissible substitutions. The latter are in fact morphisms of a category of type coercions. The relationship between the category of type coercions and the category of temporal order-sorted structures is expressed by a pair of adjoint functors (morphisms of categories). This pair of adjoint functors provides a precise characterization of the constructed initial model.

PROPOSITION 3 (Coercion category). *Let objects of* **C** *be S-sorted families of sets* $\{M_s\}_{s \in S}$ *equipped with coercion functions* $c_{s,s'}: M_s \to M_{s'}$ *whenever* $s \leqslant s'$. *Let morphisms of* **C** *be S sorted families of functions* $h_s: \{M_s\}_{s \in S} \to \{M'_s\}_{s \in S}$ *such that*

$$\begin{array}{ccc}
M_s & \xrightarrow{h_s} & M'_s \\
\downarrow{\scriptstyle c_{s,s'}} & & \downarrow{\scriptstyle c_{s,s'}} \\
M_{s'} & \xrightarrow{h_{s'}} & M'_{s'}
\end{array}$$

whenever $s \leqslant s'$. *Then* **C** *is a category.*

COROLLARY 2 (Adjoint functors). *Let* $G: \mathbf{Mod}_{\Sigma, \Pi} \to \mathbf{C}$ *be a forgetful functor. Then* G *has a left adjoint* $F: \mathbf{C} \to \mathbf{T}_{\Sigma, \Pi}$.

Proof 3.

- The forgetful functor G maps (M_S, Σ, Π) to M_S (forgets the (Σ, Π) structure) and it maps a (Σ, Π) morphism $h: (M_S, \Sigma, \Pi) \to (M'_S, \Sigma, \Pi)$ to a substitution $h: M_S \to M'_S$. Verification of functoriality is immediate. Indeed, G maps the identity morphism to the identity substitution. Furthermore, if $h: (M_S, \Sigma, \Pi) \to (M'_S, \Sigma, \Pi)$ and $g: (M'_S, \Sigma, \Pi) \to (M''_S, \Sigma, \Pi)$ are (Σ, Π) morphisms, then we have $G(hg) = G(h)\, G(g)$.

- The functor $F: \mathbf{C} \to \mathbf{Mod}_{\Sigma, \Pi}$ is defined on objects as $F(M) = \mathscr{T}_{\Sigma, \Pi}(M)$. If $h: M \to M'$ is an assignment, then $F(h) = h^*$ with h^* as constructed in Theorem 1. Furthermore, we have $F(1_M) = 1_{F(M)}$ and if $h: M \to M'$ and $g: M' \to M''$ are assignments, we have $F(hg) = F(h)\, F(g)$.

- To verify that F is a left adjoint to G [36] we have to show that the embedding $\eta_M: M \to \mathscr{T}_{\Sigma, \Pi}(M)$ is in fact a natural transformation $\eta: I \to GF$. This condition [36] requires commutativity of the diagram

$$\begin{array}{ccc} M_s & \xrightarrow{\eta_{M_s}} & GF(M_s) \\ {\scriptstyle c_{s,s'}}\downarrow & & \downarrow{\scriptstyle GF(c_{s,s'})} \\ M_{s'} & \xrightarrow{\eta_{M_{s'}}} & GF(M_{s'}) \end{array}$$

for every coercion $c_{s, s'}: M_s \to M_{s'}$.

4. TEMPORAL CONSTRAINT SYSTEM

In this section we specify the syntax of temporal constraints. The specification provides a classification of constraints according to well-established criteria. This way the expressiveness of the constraint language can be judged by comparison with other approaches based on assertions [35]. This section also provides a minimal set of formal derivation rules for temporal constraints required for the purposes of this paper. Properties of these rules needed in the further development are also established in this section.

4.1. Temporal Order-Sorted Constraints

DEFINITION 9. For $\Sigma(\mathscr{X}), \Pi$ a temporal order-sorted signature, we define:

1. $\Sigma(\mathscr{X}), \Pi$*-atoms*, or just atoms, are expressions $A(t_1, ..., t_n)$ such that $t_1, ..., t_n$ are $\Sigma(\mathscr{X})$-terms as defined in 3.5, and there is a $w = s_1 s_2 \ldots s_n$ with $A \in \Pi_w$ such that t_j has sort s_j for $j = 1, 2, ..., n$.

2. $\Sigma(\mathcal{X})$, Π-*next-atomic formulas*, or just next atoms, are expressions of the form $\bigcirc^k A(t_1, ..., t_n)$, where $A(t_1, ..., t_n)$ is a $\Sigma(\mathcal{X})$, Π- atom and $k \in N_0$. For $k = 0$, a next-atom is an atom.

DEFINITION 10. A $\Sigma(\mathcal{X})$, Π *temporal constraint* is determined by the following rules:

- *Initial constraints* specify observers of a newly created object. Initial constraints have the form

  — $self.p(A_1, A_2, ..., A_n) \leftarrow$,

where p is an observer and $A_1, A_2, ..., A_n$ are ground (i.e., variable free) terms.

- *Class invariants* are constraints that hold in all object states. Because of that class invariants require the *always* operator, do not involve the *next time* operator, and are based on observers only. A class invariant thus has the form

  — $\Box(self.p(A_1, A_2, ..., A_{np}) \leftarrow$
  $self.p_1(A_{11}, A_{12}, ..., A_{1n1}), self.p_2(A_{21}, A_{22}, ..., A_{2n2}), ...,$
  $self.p_n(A_{n1}, A_{n2}, ..., A_{nn}))$,

where $p, p_1, p_2, ..., p_n$ are observers with possibly $p_i = p$ for some p.

- *History properties* are constraints that define permissible sequences of object states. A history property has one of the following two forms,

  — $\Box(\bigcirc self.p(A_1, A_2, ..., A_n) \leftarrow$
  $self.p_1(A_{11}, A_{12}, ..., A_{1n1}), self.p_2(A_{21}, A_{22}, ..., A_{2n2}), ...,$
  $self.p_n(A_{n1}, A_{n2}, ..., A_{nn}))$
  — $\Box self.p(A_1, A_2, ..., A_n) \leftarrow \Diamond self.p(A_1, A_2, ..., A_n)$,

where $p, p_1, p_2, ..., p_n$ are observers with possibly $p_i = p$ for some i. The second form specifies a rigid (state independent) property.

- *Preconditions* are constraints that apply to mutators only. Such a constraint specifies an observer of the object state required in order to execute a mutator message

  — $\Box(self.p(A_1, A_2, ..., A_n) \leftarrow self.m(A_{m1}, A_{m2}, ..., A_{mn}))$,

where p is an observer and m is a mutator.

- Symmetrically, *postconditions* specify observers of the object state after execution of a mutator message

  — $\Box(\bigcirc self.p(A_1, A_2, ..., A_n) \leftarrow self.m(A_{m1}, A_{m2}, ..., A_{mn}))$,

where p is an observer and m is a mutator.

- *Transition constraints* specify state transitions invoked by mutator messages, and thus have the form

  — $\Box(\bigcirc self.p(A_1, A_2, ..., A_n) \leftarrow$
  $self.p_1(A_{11}, A_{12}, ..., A_{1n1}), ..., self.p_n(A_{n1}, A_{n2}, ..., A_{nn}),$
  $self.m(A_{m1}, A_{m2}, ..., A_{mn}))$,

where $p, p_1, p_2, ..., p_n$ are observers with possibly $p_i = p$ for some i, and m is a mutator.

• *Constructor constraints* specify the observers of an object created as the result of execution of a constructor message

— $\square(self.f(A_{f1}, A_{f2}, ..., A_{fn}).p(A_1, A_2, ..., A_n) \leftarrow$
$self.p_1(A_{11}, A_{12}, ..., A_{1n1}), self.p_2(A_{21}, A_{22}, ..., A_{2n2}), ...,$
$self.p_n(A_{n1}, A_{n2}, ..., A_{nn}))$,

where f is a constructor and $p, p_1, p_2, ..., p_n$ are observers with possibly $p_i = p$ for some p.

All the variables appearing in the above constraints are universally quantified.

In the above classification we adopted a terminology based on [35]. Although a more general form of constraints would still admit a formal semantic model and an execution model [5], the above limited form allows code generation for methods from temporal constraints and a truly object-oriented implementation model with dynamic binding [8].

DEFINITION 11. A temporal constraint is said to be *temporally ground* if $\bigcirc$ is the only temporal operator that appears in it.

PROPOSITION 4 (Temporally ground form). *Temporal constraints as specified in Definition* 10 *may be transformed into one of the following two temporally ground forms*,

- $A \leftarrow$
- $\bigcirc^{k'} A \leftarrow \bigcirc^k B_1, ..., \bigcirc^k B_m$

where either $k' = k$ *or* $k' = succ(k)$ *and* $A, B_1, ..., B_m$ *are atoms.*

In view of the above observation [5, 10] our further investigations will concentrate on temporally ground formulas.

4.2. Rules of Deduction

Given a temporal order-sorted signature Σ, Π and a set $\mathscr{C}$ of temporal order-sorted Σ, Π-Horn clauses, the following are the rules for *deriving next-atomic temporal formulas*:

Rules 2 (Basic rules)

1. The substitution axiom is:

• $(\forall X) \bigcirc A(t_1, t_2, ..., t_n) \leftrightarrow (\forall Y) \bigcirc A(\theta^*(t_1), \theta^*(t_2), ..., \theta^*(t_n))$ for any substitution $\theta: \mathscr{X} \to \mathscr{T}_\Sigma(\mathscr{Y})$ with θ^* defined as the unique extension of θ according to Theorem 1.

2. The *next time* axiom is:

• (Next) $(\forall X) \bigcirc A \leftrightarrow \bigcirc(\forall X) A$

3. A temporal generalization of a (classical) *derivation rule* is:

• (MP) Modus Ponens: If $(\forall X)(\bigcirc^{k'} A \leftarrow \bigcirc^k B_1, ..., \bigcirc^k B_m)$ is in $\mathscr{C}$ and if $\theta: \mathscr{X} \to \mathscr{T}_\Sigma(\mathscr{Y})$ is a substitution such that for each B_i in the body of the clause the next-atomic sentence $(\forall Y) \bigcirc^k \theta B_i$ is derivable, then so is $(\forall Y) \bigcirc^{k'} \theta A$.

PROPOSITION 5 (Derived rules). *The following rules may be derived from the basic rules*:

- (*Next*) $(\forall X)\ \bigcirc^k A \leftrightarrow \bigcirc^k (\forall X)\ A$
- $(\forall X)\ \bigcirc^k A(t_1, t_2, ..., t_n) \leftrightarrow (\forall \mathcal{Y})\ \bigcirc^k A(\theta^*(t_1), \theta^*(t_2), ..., \theta^*(t_n))$ *for any substitution* $\theta: \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{Y})$ *with* θ^* *defined as the unique extension of* θ *according to Theorem* 1.

Proof 4. Both proofs are by induction on k. For $k = 0$ (initial step) we have the basic rules.

- Suppose (inductive hypothesis) that $(\forall X)\ \bigcirc^k A \leftrightarrow \bigcirc^k (\forall X)\ A$ holds. We then have $(\forall X)\ \bigcirc^{k+1} A \leftrightarrow (\forall X)\ \bigcirc \bigcirc^k A \leftrightarrow \bigcirc (\forall X)\ \bigcirc^k A \leftrightarrow \bigcirc \bigcirc^k (\forall X)\ A \leftrightarrow \bigcirc^{k+1} (\forall X)\ A$
- Assume (inductive hypothesis) that $(\forall X)\ \bigcirc^k A(t_1, t_2, ..., t_n) \leftrightarrow (\forall \mathcal{Y})\ \bigcirc^k A(\theta^*(t_1), \theta^*(t_2), ..., \theta^*(t_n))$. Then we have
  $(\forall X)\ \bigcirc^{k+1} A(t_1, t_2, ..., t_n) \leftrightarrow$
  $(\forall X)\ \bigcirc \bigcirc^k A(t_1, t_2, ..., t_n) \leftrightarrow \bigcirc (\forall X)\ \bigcirc^k A(t_1, t_2, ..., t_n) \leftrightarrow$
  $\bigcirc (\forall Y)\ \bigcirc^k A(\theta^* t_1, \theta^* t_2, ..., \theta^* t_n) \leftrightarrow\ \bigcirc \bigcirc^k (\forall Y)\ A(\theta^* t_1, \theta^* t_2, ..., \theta^* t_n) \leftrightarrow$
  $\bigcirc^{k+1} (\forall Y)\ A(\theta^* t_1, \theta^* t_2, ..., \theta^* t_n)$.

5. MODEL THEORY

The core results on the model theory of the temporal object-oriented constraint language *MyT* are presented in this section and in Section 6. Two categories play an important role in this development. One of them is the category of temporal order-sorted structures introduced in Section 3. The other category is determined by a set of temporal constraints. Objects of this latter category are temporal order-sorted structures satisfying a given set of constraints. The relationship between these two categories is characterized by a pair of adjoint functors. A distinguished model for a given set of constraints is constructed as a colimit of a functor which reflects the temporal nature of the paradigm. This colimit construction is a temporal generalization of the classical idea of the initial model semantics.

We first define the notion of a model of a given set of temporal constraints.

DEFINITION 12. Let (Σ, Π) be an order-sorted signature with predicates and $M = (M_S, \Sigma, \Pi)$ a Σ, Π order-sorted temporal structure. Let $A = A(t_1, ..., t_n)$ and $B_j = B_j(t_{j1}, ..., t_{jn_j})$ for $j = 1, ..., m$.

1. We say that M *satisfies* a $\Sigma(\mathcal{X}), \Pi$-temporal Horn clause $(\forall X)\ \bigcirc^{k'} A \leftarrow \bigcirc^k B_1, ..., \bigcirc^k B_m$ if for any assignment of variables $\beta: \mathcal{X} \to M$ such that $(\beta^*(t_{j1}), ..., \beta^*(t_{jn_j})) \in M^{wj}_{B_{jk}}$ for $j = 1, ..., m$, then also $(\beta^*(t_1), ..., \beta^*(t_n)) \in M^w_{A_{k'}}$.

2. For $\mathscr{C}$ a set of temporal Horn clauses, we say that a structure M *satisfies* $\mathscr{C}$ ($M \models \mathscr{C}$) iff it satisfies each clause in $\mathscr{C}$.

3. A Σ, Π order-sorted temporal structure that satisfies $\mathscr{C}$ is called a $\Sigma, \Pi, \mathscr{C}$-temporal order-sorted structure.

Temporal models as defined above are naturally objects of a category whose morphisms are defined as follows:

DEFINITION 13 ($\mathscr{C}$ morphisms). Let $M=(M_S, \Sigma, \Pi, \mathscr{C})$ and $M'=(M'_S, \Sigma, \Pi, \mathscr{C})$ be $(\Sigma, \Pi, \mathscr{C})$ order-sorted temporal structures. A (Σ, Π) morphism $h: M \to M'$ is a $(\Sigma, \Pi, \mathscr{C})$ morphism iff whenever $M=(M_S, \Sigma, \Pi, \mathscr{C})$ satisfies a $\mathscr{C}$ clause $(\forall X)\ \bigcirc^{k'} A \leftarrow \bigcirc^{k} B_1, ..., \bigcirc^{k} B_m$ for some assignment of variables $\beta: \mathscr{X} \to M$, then $M'=(M'_S, \Sigma, \Pi, \mathscr{C})$ satisfies the same clause for the assignment $h\beta$.

LEMMA 1 ($\mathscr{C}$ morphisms). *(Σ, Π) morphisms are in fact $(\Sigma, \Pi, \mathscr{C})$ morphisms.*

Proof 5. Let $M=(M_S, \Sigma, \Pi, \mathscr{C})$ and $M'=(M'_S, \Sigma, \Pi, \mathscr{C})$ be $(\Sigma, \Pi, \mathscr{C})$ order-sorted temporal structures and $h: M \to M'$ a (Σ, Π) morphism. We first have to verify that $h\beta$ is indeed an assignment of variables. This follows from the following two diagrams:

$$\begin{array}{ccccc} \mathscr{X}_s & \xrightarrow{\beta_s} & M_s & \xrightarrow{h_s} & M'_s \\ \downarrow{\scriptstyle c_{s,s'}} & & \downarrow{\scriptstyle c_{s,s'}} & & \downarrow{\scriptstyle c_{s,s'}} \\ \mathscr{X}_{s'} & \xrightarrow{\beta_{s'}} & M_{s'} & \xrightarrow{h_{s'}} & M'_{s'} \end{array}$$

The diagram on the left commutes since β is an assignment, and the diagram on the right commutes because h is a morphism.

Suppose now that $M=(M_S, \Sigma, \Pi, \mathscr{C})$ satisfies a clause $(\forall X)\ \bigcirc^{k'} A \leftarrow \bigcirc^{k} B_1, ..., \bigcirc^{k} B_m$ in $\mathscr{C}$ for some assignment $\beta: \mathscr{X} \to M$. Then $(\beta^*(t_{j1}), ..., \beta^*(t_{jn_j})) \in M^{wj}_{B_{jk}}$ implies $(h\beta^*(t_{j1}), ..., h\beta^*(t_{jn_j})) \in M'^{wj}_{B_{jk}}$ for $j=1, ..., m$ since h is a (Σ, Π) morphism. For the same reason we also have $(\beta^*(t_1), ..., \beta^*(t_n)) \in M^{w}_{A_{k'}}$ implies $(h\beta^*(t_1), ..., h\beta^*(t_n)) \in M'^{w}_{A_{k'}}$. Thus indeed $M'=(M'_S, \Sigma, \Pi, \mathscr{C})$ satisfies the $\mathscr{C}$ clause $(\forall X)\ \bigcirc^{k'} A \leftarrow \bigcirc^{k} B_1, ..., \bigcirc^{k} B_m$ for the assignment $h\beta$. This completes the proof.

PROPOSITION 6 (Full subcategory). *$\Sigma, \Pi, \mathscr{C}$-temporal order sorted structures together with Σ, Π morphisms form a category, denoted* $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$. $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ *is a full subcategory of* $\mathbf{Mod}_{\Sigma, \Pi}$

Proof 6. Starting with $\mathbf{Mod}_{\Sigma, \Pi}$, we select those Σ, Π temporal structures that satisfy $\mathscr{C}$. According to the lemma, all (Σ, Π) morphisms are in fact $(\Sigma, \Pi, \mathscr{C})$ morphisms. The identity function is obviously a $(\Sigma, \Pi, \mathscr{C})$ morphism. Furthermore, the composition of $(\Sigma, \Pi, \mathscr{C})$ morphisms is obviously a $(\Sigma, \Pi, \mathscr{C})$ morphism. Also, this composition is obviously associative. This completes the proof.

In order to get a full subcategory $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ of $\mathbf{Mod}_{\Sigma, \Pi}$, it suffices to choose, among all $\mathbf{Mod}_{\Sigma, \Pi}$ temporal structures, those that satisfy $\mathscr{C}$. The morphisms of $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ will be all the morphisms of $\mathbf{Mod}_{\Sigma, \Pi}$ among the chosen $\Sigma, \Pi, \mathscr{C}$ temporal structures.

In order to simplify notation, $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ will sometimes be denoted just as $\mathbf{Mod}_{\mathscr{C}}$.

We now proceed with the construction of the colimit model.

DEFINITION 14 (A sequence of models).

1. Let $B_{\Sigma, \Pi}=\{\bigcirc^{k} B(t_1, t_2, ..., t_n) \mid B \in \Pi_w,\ w=s_1 s_2 \ldots s_n,\ t_i \in \mathscr{T}_{\Sigma, si},$ for $i=1, 2, ..., n, k \in N_0\}$. Define $F_{\mathscr{C}}(I)=\{\bigcirc^{k'} B\theta \mid \theta: \mathscr{X} \to \mathscr{T}_{\Sigma}$ (a substitution), $(\forall X)$

$(\bigcirc^{k'} B \leftarrow \bigcirc^{k} B_1,\ \bigcirc^{k} B_2,\ ...,\ \bigcirc^{k} B_m) \in \mathscr{C},\ \bigcirc^{k} B_j \theta \in I$ for $j = 1, 2, ..., m\}$. We thus have $F^0_{\mathscr{C}}(\varnothing) = \{B\theta \mid (\forall X)(B \leftarrow) \in \mathscr{C}\}$, and $F^{i+1}_{\mathscr{C}}(\varnothing) = F_{\mathscr{C}}(F^i_{\mathscr{C}}(\varnothing))$.

2. Define a sequence of temporal structures $\mathscr{T}_{\Sigma, \Pi, \mathscr{C} i}$ as follows. The carrier of $\mathscr{T}_{\Sigma, \Pi, \mathscr{C} i}$ of sort s is $\mathscr{T}_{\Sigma s}$. For each $A \in \Pi_w$, $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}_i}$ has a relation $\mathscr{T}^{\,w}_{\Sigma, Ak} = \{(t_1, t_2, ..., t_n) \mid \bigcirc^{k} A(t_1, t_2, ..., t_n) \in F^i_{\mathscr{C}}(\varnothing), k \leqslant i\}$.

3. Similarly, define $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ as follows. The carrier of $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ of sort s is $\mathscr{T}_{\Sigma s}$. For each $A \in \Pi_w$, $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ has a relation $\mathscr{T}^{\,w}_{\Sigma, Ak} = \{(t_1, t_2, ..., t_n) \mid \bigcirc^{k} A(t_1, t_2, ..., t_n) \in \bigcup_{i=0}^{\infty} F^i_{\mathscr{C}}(\varnothing)\}$.

THEOREM 2 ($\mathscr{C}$ model). *$\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ is a model for $\mathscr{C}$.*

Proof 7. All that is needed is to prove that the predicate condition is satisfied. We thus have to prove the following property for every $\mathscr{C}$ clause $(\forall X) \bigcirc^{k'} A \leftarrow \bigcirc^{k} B_1, ..., \bigcirc^{k} B_m$: Let $A = A(t_1, ..., t_n)$ and $B_j = B_j(t_{j1}, ..., t_{jn_j})$ for $j = 1, ..., m$ with $\beta: \mathscr{X} \to \mathscr{T}_{\Sigma}$ an assignment of variables. If $(\beta^*(t_{j1}), ..., \beta^*(t_{jn_j})) \in \mathscr{T}^{\,wj}_{B_{jk}}$ for $j = 1, ..., m$, then also $(\beta^*(t_1), ..., \beta^*(t_n)) \in \mathscr{T}^{\,w}_{A_{k'}}$.

Let $(\beta^*(t_{j1}), ..., \beta^*(t_{jn_j})) \in \mathscr{T}^{\,wj}_{B_{jk}}$ for $j = 1, ..., m$. Then $\bigcirc^{k} B_j(\beta^*(t_{j1}), ..., \beta^*(t_{jn_j})) \in F^i_{\mathscr{C}}(\varnothing)$ for some $i, j = 1, 2, ..., m$. Since $(\forall X)(\bigcirc^{k'} A \leftarrow \bigcirc^{k} B_1, ..., \bigcirc^{k} B_m)$ is a clause in $\mathscr{C}$, we have $\bigcirc^{k'} A(\beta^*(t_1), ..., \beta^*(t_n)) \in F^{i+1}_{\mathscr{C}}(\varnothing)$. But then $(\beta^*(t_1), ..., \beta^*(t_n)) \in \mathscr{T}^{\,w}_{A_{k'}}$ in $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$.

THEOREM 3 (Colimit model).

1. *$\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ is equipped with a family of morphisms $\mu_i: \mathscr{T}_{\Sigma, \Pi, \mathscr{C} i} \to \mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ such that the following diagram commutes*:

$$\begin{array}{ccc} \mathscr{T}_{\Sigma, \Pi, \mathscr{C} i} & \xrightarrow{F^{i, i+1}_{\mathscr{C}}} & \mathscr{T}_{\Sigma, \Pi, \mathscr{C} i+1} \\ \Big\downarrow{\scriptstyle \mu_i} & & \Big\downarrow{\scriptstyle \mu_{i+1}} \\ \mathscr{T}_{\Sigma, \Pi, \mathscr{C}} & \xrightarrow{=} & \mathscr{T}_{\Sigma, \Pi, \mathscr{C}} \end{array}$$

2. *Given a temporal structure $M_{\Sigma, \Pi, \mathscr{C}}$ and a family of morphisms $\tau_i: \mathscr{T}_{\Sigma, \Pi, \mathscr{C}_i} \to M_{\Sigma, \Pi, \mathscr{C}}$ such that*

$$\begin{array}{ccc} \mathscr{T}_{\Sigma, \Pi, \mathscr{C}_i} & \xrightarrow{F^{i, i+1}_{\mathscr{C}}} & \mathscr{T}_{\Sigma, \Pi, \mathscr{C} i+1} \\ \Big\downarrow{\scriptstyle \tau_i} & & \Big\downarrow{\scriptstyle \tau_{i+1}} \\ M_{\Sigma, \Pi, \mathscr{C}} & \xrightarrow{=} & M_{\Sigma, \Pi, \mathscr{C}} \end{array}$$

there exists a unique morphism $\tau: \mathscr{T}_{\Sigma, \Pi, \mathscr{C}} \to M_{\Sigma, \Pi, \mathscr{C}}$ such that the following diagram commutes for all i:

$$\begin{array}{ccc} \mathscr{T}_{\Sigma, \Pi, \mathscr{C} i} & \xrightarrow{\mu_i} & \mathscr{T}_{\Sigma, \Pi, \mathscr{C}} \\ \Big\downarrow{\scriptstyle =} & & \Big\downarrow{\scriptstyle \tau} \\ \mathscr{T}_{\Sigma, \Pi, \mathscr{C}_i} & \xrightarrow{\tau_i} & M_{\Sigma, \Pi, \mathscr{C}} \end{array}$$

Proof 8. 1. Each μ_i and $F^{i,i+1}_{\mathscr{C}}$ are defined as the identity functions for each sort s. Thus the satisfaction of the morphism and the coercion conditions is immediate, as is the commutativity of the diagram. What remains to be checked is the predicate condition.

Let $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma, A}$ in $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}i}$. But then $\bigcirc^k A(t_1, t_2, ..., t_n) \in F^{i}_{\mathscr{C}}(\varnothing)$ for some i where $k \leqslant i$. This implies that both $\bigcirc^k A(t_1, t_2, ..., t_n) \in F^{i+1}_{\mathscr{C}}(\varnothing)$ and $\bigcirc^k A(t_1, t_2, ..., t_n) \in \bigcup_{i=0}^{\infty} F^{i}_{\mathscr{C}}(\varnothing)$. Thus $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma, A}$ in $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}i+1}$ and $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma, A}$ in $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$.

2. Since each $F^{i,i+1}_{\mathscr{C}}$ is the identity function, we have that $\tau_i = \tau_{i+1}$ for all i. Thus $\tau = \tau_i$ for any i.

COROLLARY 3 (Colimit property). *Let ω be a category generated by the graph $0 \to 1 \to 2 \to \ldots$, $i \to succ(i), \ldots$. Define a functor $F: \omega \to \mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ such that $F(i) = \mathscr{T}_{\Sigma, \Pi, \mathscr{C}i}$. Then $\mathscr{T}_{\Sigma, \Pi, \mathscr{C}}$ is the colimit of the functor F.*

When we forget about the temporal constraints, the above construction reduces to the the construction of the free temporal (Σ, Π) structure from Theorem 1.

COROLLARY 4 (Forgetful functor). *Let $G: \mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}} \to \mathbf{Mod}_{\Sigma, \Pi}$ be a forgetful functor which maps a temporal structure $(M_S, \Sigma, \Pi, \mathscr{C})$ to the temporal structure (M_S, Σ, Π). Then the composite functor $GF: \omega \to \mathbf{T}_{\Sigma, \Pi}$ has a colimit, which is precisely the initial temporal structure $\mathscr{T}_{\Sigma}$ from Corollary* 1.

The presented colimit construction allows the following characterization of the relationship between the categories $\mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$ and $\mathbf{Mod}_{\Sigma, \Pi}$:

THEOREM 4 (Adjoint functors). *The forgetful functor $G: \mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}} \to \mathbf{Mod}_{\Sigma, \Pi}$ has a left adjoint $L: \mathbf{Mod}_{\Sigma, \Pi} \to \mathbf{Mod}_{\Sigma, \Pi, \mathscr{C}}$.*

Proof 9. Let $\mathscr{C}' = \mathscr{C} \cup \{\bigcirc^k B(m_1, m_2, ..., m_n) \leftarrow | (m_1, m_2, ..., m_n) \in M^{w}_{Bk}\}$. Define L on objects as $L: (M_S, \Sigma, \Pi) \to \mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}'}$. Note that since $\mathscr{C} \subset \mathscr{C}'$, a $(\Sigma, \Pi, \mathscr{C}')$ structure is necessarily a $(\Sigma, \Pi, \mathscr{C})$ structure, and the same is true for morphisms.

We have to prove that any (Σ, Π) morphism $h: (M_S, \Sigma, \Pi) \to (M'_S, \Sigma, \Pi)$ extends to a unique $(\Sigma, \Pi, \mathscr{C}')$ morphism $h_{\mathscr{C}'}: \mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}'} \to (M'_S, \Sigma, \Pi)$ such that the following diagram commutes:

$$\begin{array}{ccc} (M_S, \Sigma, \Pi) & \xrightarrow{\eta_M} & \mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}'} \\ = \downarrow & & \downarrow h_{\mathscr{C}'} \\ (M_S, \Sigma, \Pi) & \xrightarrow{h} & (M', \Sigma, \Pi) \end{array}$$

where η_M is an embedding.[2]

Note that the above diagram consists of morphisms in $\mathbf{Mod}_{\Sigma, \Pi}$. Define $h_{\mathscr{C}'}$ as follows:

- $h_{\mathscr{C}'}(m) = h(m)$ for $m \in M$
- $h_{\mathscr{C}'}(\sigma(t_1, t_2, ..., t_n)) = f_{\sigma}(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n))$.

[2] In fact, $\eta: I \to GL$ is a natural transformation with I the identity functor.

This makes $h_{\mathscr{C}'}$ a Σ-morphism. In addition, according to Theorem 1, $h_{\mathscr{C}'}$ satisfies the coercion condition. It remains to prove that $h_{\mathscr{C}'}$ is in fact a (Σ, Π) morphism. Then according to Lemma 1, $h_{\mathscr{C}'}$ will be a $\mathscr{C}'$ morphism.

In order to verify the predicate condition assume that $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma(M),\, Ak}$. We then also have to show that $(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \in M'^{w}_{Ak}$. Note that $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma(M),\, Ak}$ implies $\bigcirc^k A(t_1, t_2, ..., t_n) \in F^i_{\mathscr{C}'}(\varnothing)$ for some i where $k \leqslant i$. The proof proceeds by induction on i.

- Let $\bigcirc^k A(t_1, t_2, ..., t_n) \in F^i_{\mathscr{C}'}(\varnothing)$ for $i = 0$. Then $k = 0$ and $A(t_1, t_2, ..., t_n) \leftarrow$ is a ground $\mathscr{C}'$ clause. This means that either $(t_1, t_2, ..., t_n) \in M^w_A$ or $A(t_1, t_2, ..., t_n) \leftarrow$ is a ground $\mathscr{C}$ clause. If $(t_1, t_2, ..., t_n) \in M^w_A$ then also $(h(t_1), h(t_2), ..., h(t_n)) \in M'^{w}_A$ since h is a (Σ, Π) morphism. By the definition of $h_{\mathscr{C}'}$ this implies $(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \in M'^{w}_A$. If, on the other hand, $A(t_1, t_2, ..., t_n) \leftarrow$ is a ground $\mathscr{C}$ clause, then $A(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \leftarrow$ is a ground $\mathscr{C}'$ clause. This follows from the fact that if h' is a morphism, it is then also a valid assignment.

But now (M'_S, Σ, Π) is a $(\Sigma, \Pi, \mathscr{C}')$ structure, and it thus satisfies this clause.

2. The inductive hypothesis is that if for some i, $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma(M),\, Bk}$ with $k \leqslant i$, then also $(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \in M'^{w}_{Bk}$ for all $B \in \Pi$. Note that $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma(M),\, B_k}$ with $k \leqslant i$ means $\bigcirc^k B(t_1, t_2, ..., t_n) \in F^i_{\mathscr{C}'}(\varnothing)$.

3. Let $(t_1, t_2, ..., t_n) \in \mathscr{T}^{w}_{\Sigma(M),\, Ak'}$ where $k' \leqslant i+1$. This means $\bigcirc^{k'} A(t_1, t_2, ..., t_n) \in F^{i+1}_{\mathscr{C}'}(\varnothing)$. According to the construction of $\mathscr{T}_{\Sigma(M),\, \Pi,\, \mathscr{C}'}$, it must then be possible to derive from $\mathscr{C}'$ a ground clause $\bigcirc^{k'} A(t_1, t_2, ..., t_n) \leftarrow \bigcirc^k B_1(t_{11}, t_{12}, ..., t_{1n_1}), ..., \bigcirc^k B_m(t_{m1}, t_{m2}, ..., t_{mn_m})$ such that $\bigcirc^k B_j(t_{j1}, t_{j2}, ..., t_{jn_j}) \in F^i_{\mathscr{C}'}(\varnothing)$ for $j = 1, 2, ..., m$. This means $(t_{j1}, t_{j2}, ..., t_{jn}) \in \mathscr{T}^{w}_{\Sigma(M),\, B_j k}$, which by the inductive hypothesis implies $(h_{\mathscr{C}'}(t_{j1}), h_{\mathscr{C}'}(t_{j2}), ..., h_{\mathscr{C}'}(t_{jn_j})) \in M'^{w}_{B_j k}$. But (M', Σ, Π) satisfies the ground $\mathscr{C}'$ clause $\bigcirc^{k'} A(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \leftarrow \bigcirc^k B_1(h_{\mathscr{C}'}(t_{11}), h_{\mathscr{C}'}(t_{12}), ..., h_{\mathscr{C}'}(t_{1n_1})), ..., \bigcirc^k B_m(h_{\mathscr{C}'}(t_{m1}), h_{\mathscr{C}'}(t_{m2}), ..., h_{\mathscr{C}'}(t_{mn_m}))$ because (M', Σ, Π) is a $(\Sigma, \Pi, \mathscr{C}')$ structure, and thus $(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n)) \in M'^{w}_{Ak'}$.

6. TEMPORAL THEORIES

In this section we study the relationships among different classes created by various, specific forms of inheritance. Each temporal class is naturally viewed as a temporal theory. The relationships among classes are represented by suitably defined morphisms.

Consider the relationship between temporal classes *MovingObject* and *Satellite*. Because of the change of interpretation of *MyClass* in the class *Satellite*, the inherited order-sorted signature of *Satellite* is different from the underlying order-sorted signature of *MovingObject*. The temporal constraints in the class *Satellite* inherited from the class *MovingObject* must also be reinterpreted accordingly. The transformation of the underlying order-sorted signatures will be shown to correspond to a morphism of a suitably defined category of order-sorted signatures. The overall transformation of the underlying order-sorted signature and the associated temporal constraints will be shown to correspond to a morphism of a suitably defined category of temporal theories.

6.1. Signatures and their Morphisms

DEFINITION 15 (Signature morphism). A signature morphism $\phi: (S, \leqslant, \Sigma, \Pi) \to (S', \leqslant', \Sigma', \Pi')$ consists of the following monotone functions:

1. A function $f: S \to S'$
2. A family of functions $g: \Sigma_{w,s} \to \Sigma'_{f(w),f(s)}$
3. A family of functions $p: \Pi_w \to \Pi'_{f(w)}$.

PROPOSITION 7 (Signature category). *Order-sorted signatures with predicates and their morphisms form a category, denoted* **Sig**.

EXAMPLE 8 (Signature morphism).

$S = \{Number, MovingObject\}$
$\leqslant = \{$only reflexive relationships$\}$
$\Sigma = \{+, *: Number, Number \to Number, ..., clone: MovingObject \to MovingObject\}$
$\Pi = \{x, y, z: MovingObject, Number, speed, heading, pitch: MovingObject, Number\}$

$S' = \{Number, Satellite\}$
$\leqslant = \{$only reflexive relationships$\}$
$\Sigma' = \{+, *: Number, Number \to Number, ..., clone: Satellite \to Satellite\}$
$\Pi' = \{x, y, z: Satellite, Number, speed, heading, pitch: Satellite, Number$
$axisX, axisY: Satellite, Number\}$

$f(Number) = Number$, $f(MovingObject) = Satellite$
$g(+) = +$, $g(*) = *$, ...,
$g(clone) = clone: Satellite \to Satellite$
$p(x) = x: Satellite, Number$, $p(y) = y: Satellite, Number$,
$p(z) = z: Satellite, Number$,
$p(speed) = speed: Satellite, Number$, $p(pitch) = pitch: Satellite, Number$,
$p(heading) = heading: Satellite, Number$,
$p(axisX) = axisX: Satellite, Number$, $p(axisY) = axisY: Satellite, Number\}$

EXAMPLE 9. (Signature morphism).
This is an example of a signature morphism which is in fact an embedding.

$(S, \leqslant, \Sigma, \Pi\}$ as in Example 8. $S'' = \{Number, MovingObject, Satellite\}$
$\leqslant'' = \{Satellite \leqslant MovingObject$ plus reflexive relationships $\}$
$\Sigma'' = \Sigma \cup \Sigma'$, $\Pi'' = \Pi \cup \Pi'$, where Σ' and Π' are as in the Example 8.
$f: S \to S''$ defined as follows:
$f(Number) = Number$, $f(MovingObject) = MovingObject$,
and the functions g and p are defined accordingly in the obvious way.

6.2. Temporal theories

DEFINITION 16 (Temporal theory). A temporal theory T is a tuple $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ where $\mathscr{C}$ is a set of temporal (Σ, Π) Horn clauses.

DEFINITION 17 (Theory morphism). Let $T = (S, \leqslant, \Sigma, \Pi, \mathscr{C})$ be a temporal theory and $M = (M_S, \Sigma, \Pi, \mathscr{C})$ a model for the theory T, denoted $M \models \mathscr{C}$.

Let $\mathscr{C}^*$ be the least set defined as $\mathscr{C}^* = \{\zeta \mid M \models \mathscr{C}$ *implies* $M \models \zeta\}$ where ζ denotes a (Σ, Π) temporal constraint.

A theory morphism $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ is a signature morphism $(S, \leqslant, \Sigma, \Pi) \to (S', \leqslant', \Sigma', \Pi')$ such that

$\zeta \in \mathscr{C} \Rightarrow \phi(\zeta) \in \mathscr{C}'^*$.

PROPOSITION 8 (Theory category). *Temporal theories and their morphisms form a category, denoted* **Th**.

Definition 17 and Proposition 8 are in the spirit of [21].

EXAMPLE 10 (Theory morphism). The signatures and the signature morphism are as in Example 8. The temporal constraints $\mathscr{C}$ for the temporal theory $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ are given below:

((∀) *M*: *MovingObject*, (∀) *X*, (∀) *Y*, (∀) *Z*: *Number*, (∀) *S*: *Number*, (∀) *D*(∀) *G*: *Number*)
(□(○x(M, X + S*D.cos()*G.cos())
← x(M, X), speed(M, S), heading(M, G), pitch(M, D)),
□(○y(M, Y + S*D.cos()*D.sin()
← y(M, Y), speed(M, S), heading(M, G), pitch(M, D))
□(○z(M, Z + S*D.sin())
← z(M, Z), speed(M, S), pitch(M, D))
□x(clone(M, X, Y, Z), X) ←,
□y(clone(M, X, Y, Z), Y) ←,
□z(clone(M, X, Y, Z), Z) ←,
□(heading(clone(M, X, Y, Z), G) ← heading(M, G)),
□(speed(clone(M, X, Y, Z), S) ← speed(M, S)),
□(pitch(clone(M, X, Y, Z), D) ← pitch(M, D))
)

The temporal constraints $\mathscr{C}'$ of the temporal theory $(S', \leqslant', \Sigma', \Pi')$ consist of the constraints given above with *Satellite* replacing *MovingObject* plus the constraints given below:

((∀) *T*: *Satellite*, (∀) *X*, (∀) *Y*: *Number*, (∀) *Ax*, (∀) *Ay*: *Number*, (∀) *S*: *Number*)
(□z(T, 0) ←,
□pitch(T, 0) ←,
□axisX(T, Ax) ← ◇axisX(T, Ax),
□axisY(T, Ay) ← ◇axisY(T, Ay),
□speed(T, S) ← ◇speed(T, S),
□(heading(T, −arctan((X*sqr(axisY))/(Y*sqr(axisX)))
← x(T, X), y(T, Y), axisX(T, Ax), axisY(T, Ay))
)

EXAMPLE 11 (Theory morphism). The signatures and the signature morphism are as in Example 9. The constraints $\mathscr{C}''$ are constructed as $\mathscr{C} \cup \mathscr{C}'$.

6.3. Theories and orderings

Consider now a fixed set of sorts such as $S = \{Number, MovingObject, Satellite\}$. The set S may be equipped with a variety of orderings defined in Section 3.2. These orderings differ in their extensions to $S^* \times S$. We thus necessarily obtain different temporal theories for different orderings. In this section we show that some relationships among such theories are in fact morphisms of temporal theories.

PROPOSITION 9 (Theories and orderings).

1. *Let* $(S, <=, \Sigma, \Pi, \mathscr{C})$ *and* $(S, \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be temporal theories such that* $\leqslant' \in \{ \sqsubseteq, \preccurlyeq \}$ *and* $\mathscr{C} \subseteq \mathscr{C}'$. *Then the following triple is a theory morphism*:

   (a) f: $S \to S$ *is the identity function*

   (b) *A family of functions* g: $\Sigma_{w,s} \to \Sigma'_{f(w),f(s)}$ *defined as* $g(\sigma) = \sigma$

   (c) *A family of functions* p: $\Pi_w \to \Pi'_{f(w)}$ *defined as* $p(P) = P$

2. *Let* $(S, \preccurlyeq, \Sigma, \Pi, \mathscr{C})$ *and* $(S, \sqsubseteq, \Sigma', \Pi', \mathscr{C}')$ *be temporal theories. Then the triple* (f, g, p) *as defined in* 1. *above is a theory morphism.*

EXAMPLE 12 (Theories and orderings). Let $(S'', \sqsubseteq, \Sigma'', \Pi'', \mathscr{C}'')$ where $S'' = \{Number, MovingObject, Satellite\}$ and $\leqslant'' = \{Satellite \leqslant MovingObject$ plus reflexive relationships$\}$ with extension to $S^* \times S$ as defined in 3.2 and
$\Sigma'' = \Sigma \cup \Sigma'$, $\Pi'' = \Pi \cup \Pi'$
where Σ' and Π' are as in Example 9.

$(S, \preccurlyeq, \Sigma, \Pi, \mathscr{C})$ is defined as follows: $S = S'' = \{Number, MovingObject, Satellite\}$
$\preccurlyeq = \{Satellite \preccurlyeq MovingObject$ plus reflexive relationships$\}$
and extension to $S^* \times S$ as defined in 3.2.
$\Sigma = \{ +, *$: *Number*, *Number* $\to$ *Number*, ...,
clone: *MovingObject* $\to$ *MovingObject*$\}$
$\Pi = \{x, y, z$: *MovingObject*, *Number*,
speed, *heading*, *pitch*: *MovingObject*, *Number*,
axisX, *axisY*: *Satellite*, *Number*$\}$

f: $S \to S''$ is defined as an embedding, with functions g and p defined likewise as the obvious injections.

7. PRESERVATION OF BEHAVIOR

Consider temporal classes T and T' viewed as theories. Let T' be derived by inheritance from T. A fundamental question is whether the behavior of objects of a class T is preserved in the behavior of objects of a class T'. Addressing this issue in a procedural object-oriented paradigm is very difficult. This is due to the complexity of verifying assertions about procedurally decomposed methods.

The issue of preservation of behavior is expressed via substitutability. Object-oriented type systems guarantee that an object of type T' may be substituted where an object of type T is expected. If the rules of subtyping are satisfied, such substitutions will cause no run-time type errors. However, object-oriented type systems do not address the issue of the semantics of such substitutions, just the issue of the

compatibility of method signatures. Expressing these semantic conditions is beyond the expressiveness of an object-oriented type system. This is where the main advantage of a semantic, logic-based paradigm is. Such a paradigm is able to express semantic (behavioral) compatibility constraints which an object-oriented type system cannot express.

The behavioral compatibility has two aspects. The first one is mapping objects of type T' into objects of type T such that the mapped objects (images) satisfy all the constraints of T. This is the issue of the static semantic compatibility. It is expressed in the model theoretic terms by the requirement that there exists a forgetful functor U: $\mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ which maps a model of class T' to a model of class T. Of course, U is also required to correctly map morphisms and commute with their composition. In Section 7.1 we prove that the static semantic compatibility conditions are satisfied both for embedding and for constrained matching.

The static semantic compatibility conditions are only the first step in establishing behavioral compatibility. The impact of dynamic dispatch of methods requires further investigations. This is the issue of preservation of behavior under the operational model for temporal classes. The operational model for both embedding and constrained matching is described in Section 7.2. The results on the dynamic preservation of behavior are proved as well.

7.1. Static Semantics

Let $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ be a temporal theory of the class *MovingObject* and $(S', \leqslant, \Sigma', \Pi', \mathscr{C}')$ a temporal theory of the class *Satellite*. Then we have that $\Sigma \subseteq \Sigma'$, $\Pi \subseteq \Pi'$, and $\mathscr{C} \subseteq \mathscr{C}'$. This amounts to inheriting all the methods without changing their signatures and introducing some new ones. The desired model theoretic properties are obtained if no redefinition of the inherited constraints is carried out and just additional constraints are introduced. This is the case of embedding of temporal theories. We first prove model theoretic results pertinent to this case.

PROPOSITION 10 (Adjoint functors for embedding). *Let* $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ *and* $(S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be temporal theories such that* $\Sigma \subseteq \Sigma'$, $\Pi \subseteq \Pi'$ *and* $\mathscr{C} \subseteq \mathscr{C}'$. *Then there exists a forgetful functor* U: $\mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$.

Proof 10.

- The forgetful functor U is defined to map $(M_{S'}, \Sigma', \Pi', \mathscr{C}')$ to $(M_S, \Sigma, \Pi, \mathscr{C})$, where $M_S = \{M_{s'} \mid s' \in S\}$.
- If h': $M' \to K'$ is a morphism in $\mathbf{Mod}_{\mathscr{C}'}$, then we have a family of functions $h_{s'}$: $M_{S'} \to K_{S'}$. Restricting this family to the sorts in S produces a family of functions h_s: $M_S \to K_S$, so that $Uh' = h$. The family h_s has all the desired properties because $h_{s'}$ does, and $\Sigma \subseteq \Sigma'$, $\Pi \subseteq \Pi'$ and $\mathscr{C} \subseteq \mathscr{C}'$.

The proposition that follows is a generalization of Theorem 4.

THEOREM 5 (Adjoint functors for embedding). *Let* $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ *and* $(S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be temporal theories such that* $\Sigma \subseteq \Sigma'$, $\Pi \subseteq \Pi'$, *and* $\mathscr{C} \subseteq \mathscr{C}'$. *The forgetful functor* U: $\mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ *has a left adjoint* F: $\mathbf{Mod}_{\mathscr{C}} \to \mathbf{Mod}_{\mathscr{C}'}$.

Proof 11. We have to prove that for any $(\Sigma, \Pi, \mathscr{C})$ morphism $h_{\mathscr{C}}$: $(M_S, \Sigma, \Pi, \mathscr{C}) \to U(K_{S'}, \Sigma', \Pi', \mathscr{C}')$, there exists a unique $(\Sigma', \Pi', \mathscr{C}')$ morphism $h_{\mathscr{C}'}$: $F(M_S, \Sigma, \Pi, \mathscr{C}) \to (K_{S'}, \Sigma', \Pi', \mathscr{C}')$, which makes the following diagram in $\mathbf{Mod}_{\mathscr{C}}$ commute

$$\begin{array}{ccc} (M_S, \Sigma, \Pi, \mathscr{C}) & \xrightarrow{\eta_M} & \mathscr{T}_{\Sigma(M),\Pi,\mathscr{C}'} \\ = \downarrow & & \downarrow h_{\mathscr{C}'} \\ (M_S, \Sigma, \Pi, \mathscr{C}) & \xrightarrow{h_{\mathscr{C}}} & U(K_{S'}, \Sigma', \Pi', \mathscr{C}') \end{array}$$

where η: $I \to UF$ is a natural transformation.

Define F on objects as $F(M_S, \Sigma, \Pi, \mathscr{C}) = \mathscr{T}_{\Sigma(M),\Pi,\mathscr{C}'}$. $\mathscr{T}_{\Sigma(M),\Pi,\mathscr{C}'}$ becomes a Σ', Π' structure with the empty carriers for sorts in $S' \backslash S$ and empty relations for predicates in $\Pi' \backslash \Pi$. We now define $h_{\mathscr{C}'}$ to agree with $h_{\mathscr{C}}$ on Σ terms: (i) $h_{\mathscr{C}'}(m) = h_{\mathscr{C}}(m)$ for $m \in M$ and (ii) $h_{\mathscr{C}'}(\sigma(t_1, t_2, ..., t_n)) = f_\sigma(h_{\mathscr{C}'}(t_1), h_{\mathscr{C}'}(t_2), ..., h_{\mathscr{C}'}(t_n))$. This makes $h_{\mathscr{C}'}$ a Σ' morphism because $\mathscr{T}_{\Sigma(M),\Pi,\mathscr{C}'}$ has empty sets of terms for the sorts in $S' \backslash S$. Following the logic in Proof 9 we can now prove that $h_{\mathscr{C}'}$ is a (Σ', Π') morphism. According to Proposition 6, (Σ', Π') morphisms are in fact $(\Sigma', \Pi', \mathscr{C}')$ morphisms. Note that η is a natural embedding since $\Sigma \subseteq \Sigma'$ and $\Pi \subseteq \Pi'$. Furthermore, $h_{\mathscr{C}'}$ has been constructed in such a way that it agrees with $h_{\mathscr{C}}$ on Σ terms. This completes the proof.

EXAMPLE 13. A theory morphism $(S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}'')$ from Example 11 gives rise to a forgetful functor $\mathbf{Mod}_{\mathscr{C}''} \to \mathbf{Mod}_{\mathscr{C}}$.

The result specified in Proposition 10 carries over to constrained matching. The construction of the forgetful functor is more subtle, and the proof is more elaborate.

THEOREM 6 (Forgetful functor for constrained matching). *Let* ϕ: $(S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be a constrained matching morphism of temporal theories. Then there exists a forgetful functor* U: $\mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$.

Proof 12. Let C be the receiver sort in S and C' the receiver sort in S'. If $\phi = (f, g, p)$ is a constrained matching morphism, then $f(C) = C'$ and $f(s_i) = s_i$ otherwise. In fact, $S \backslash \{C\} \subseteq S'$. Furthermore, $\mathscr{C}' = \{\zeta[C/C'] \sqcup \zeta'_{add}\}$ where ζ'_{add} are the additional constraints introduced in $\mathscr{C}'$.

In addition, if $\sigma \in \Sigma_{w,s}$, then we have $\sigma \in \Sigma'_{w[C/C'],s[C/C']}$. Likewise, $A \in \Pi_w$ implies $A \in \Pi'_{w[C/C']}$.

The forgetful functor U: $\mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ which maps $(M_{S'}, \Sigma', \Pi', \mathscr{C}')$ to $(M_S, \Sigma, \Pi, \mathscr{C})$ is defined as follows:

1. A family of carriers is defined as $M_s = M_{s'}$ for $s' \in S$ and $M_C = M_{C'}$.

2. We have a family of coercion functions $M_{s_1} \to M_{s_2}$ for each $s_1 \in S$ and $s_2 \in S$ such that $s_1 \leqslant s_2$. This family is obtained from the coercion functions $M_{s'_1} \to M_{s'_2}$ by substitution of the receiver sorts $M_{s'_1[C'/C]} \to M_{s'_2[C'/C]}$.

3. For every $\sigma \in \Sigma_{w,s}$, where $w = s_1 s_2 \ldots s_n$, we have a function $f_\sigma : M_w \to M_s$. These functions are obtained for $\sigma' \in \Sigma'_{w',s'}$ such that $\sigma'[C'/C] \in \Sigma_{w,s}$, i.e., $w'[C'/C] = w$, $s'[C'/C] = s$.

4. The coercion condition required whenever $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \leqslant w_2$

$$\begin{array}{ccc} M_{w_1} & \xrightarrow{f_\sigma} & M_{s_1} \\ {\scriptstyle c_{w_1,w_2}}\downarrow & & \downarrow{\scriptstyle c_{s_1,s_2}} \\ M_{w_2} & \xrightarrow{f_\sigma} & M_{s_2} \end{array}$$

is obtained from the corresponding diagram for the models in $\mathbf{Mod}_{\mathscr{C}'}$ with $w_1[C/C'] = w'_1 \in \Sigma'_{w'_1}$, $s_1[C/C'] = s'_1$ and $w_2[C/C'] = w'_2 \in \Sigma'_{w'_2}$, $s_2[C/C'] = s'_2$

5. Let $A \in \Pi_w$. But then $A \in \Pi'_{w'}$ where $w' = w[C/C']$. We have $M^{w'}_{A_i} \subseteq M_{w'}$. But this means $M^{w}_{A_i} \subseteq M_w$ since $M_s = M_{s'}$ for $s' \in S \setminus \{C\}$ and $M_{C'} = M_C$.

Now we have to prove that U maps correctly morphisms. Let $h' : M_{\mathscr{C}'} \to K_{\mathscr{C}'}$ be a Σ', Π', $\mathscr{C}'$ morphism.

1. We have the following diagram which holds for models in $\mathbf{Mod}_{\mathscr{C}'}$:

$$\begin{array}{ccc} M_{w'} & \xrightarrow{f_\sigma} & M_{s'} \\ {\scriptstyle h'_{w'}}\downarrow & & \downarrow{\scriptstyle h'_{s'}} \\ K_{w'} & \xrightarrow{f_\sigma} & K_{s'} \end{array}$$

If $w'[C'/C] = w \in \Sigma$ and $s'[C'/C] = s \in S$, we get a diagram that holds for the models in $\mathbf{Mod}_{\mathscr{C}}$:

$$\begin{array}{ccc} M_{w} & \xrightarrow{f_\sigma} & M_{s} \\ {\scriptstyle Uh'_{w}}\downarrow & & \downarrow{\scriptstyle Uh'_{s}} \\ K_{w} & \xrightarrow{f_\sigma} & K_{s} \end{array}$$

2. Consider the coercion conditions. We have to show that we have the coercion diagram

$$\begin{array}{ccc} M_{s_1} & \xrightarrow{h_{s_1}} & K_{s_1} \\ {\scriptstyle c_{s_1,s_2}}\downarrow & & \downarrow{\scriptstyle c_{s_1,s_2}} \\ M_{s_2} & \xrightarrow{h_{s_2}} & K_{s_2} \end{array}$$

for each $s_1 \leqslant s_2$ where $h = Uh'$. But if $s_1 \in S$ and $s_2 \in S$, we have that $s_1[C/C'] \in S'$ and $s'_2[C/C'] \in S'$. We have the coercion diagram for models in $\mathbf{Mod}_{\mathscr{C}'}$ which amounts to the above required diagram for models in $\mathbf{Mod}_{\mathscr{C}}$.

3. Consider now the temporal predicate condition. We have $h'\colon M'^{w'}_{A_i} \to K'^{w'}_{A_i}$. If $w'[C'/C] = w$ we get $h\colon M^{w}_{A_i} \to K^{w}_{A_i}$ where $h = Uh'$.

Let $(a_1, a_2, ..., a_n) \in M^{w}_{A_i}$. But then $(a_1, a_2, ..., a_n) \in M'^{w'}_{A_i}$ where $w'[C'/C] = w$. This implies $(h'(a_1), h'(a_2), ..., h'(a_n)) \in K'^{w'}_{A_i}$. But then $(h(a_1), h(a_2), ..., h(a_n)) \in K^{w}_{A_i}$.

4. We still have to verify that h preserves $\mathscr{C}$ constraints, i.e., we have to show that $M \models \zeta \Rightarrow K \models \zeta$. Let $M \models \zeta$. But then $M' \models \zeta[C/C']$. This implies $K' \models \zeta[C/C']$. But then we have $K \models \zeta$.

As in Theorem 5, we can prove that the functor $U\colon \mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ for constrained matching constructed above has a left adjoint $F\colon \mathbf{Mod}_{\mathscr{C}} \to \mathbf{Mod}_{\mathscr{C}'}$.

THEOREM 7 (Adjoint functors for constrained matching). *Let* $\phi\colon (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be a constrained matching morphism of temporal theories. The forgetful functor* $U\colon \mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ *has a left adjoint* $F\colon \mathbf{Mod}_{\mathscr{C}} \to \mathbf{Mod}_{\mathscr{C}'}$.

Proof 13. This proof follows the logic of the proof of Theorem 5 with some modifications. Given a $\Sigma, \Pi, \mathscr{C}$ morphism $h_{\mathscr{C}}\colon M \to UK'$, there exists a unique $\Sigma', \Pi', \mathscr{C}'$ morphism $h_{\mathscr{C}'}\colon \mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}'} \to K'$ such that $Uh_{\mathscr{C}'} = h_{\mathscr{C}}$, as in the diagram below:

$$\begin{array}{ccc} M & \xrightarrow{\eta_M} & \mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}'} \\ = \downarrow & & \downarrow h_{\mathscr{C}'} \\ M & \xrightarrow{h_{\mathscr{C}}} & UK' \end{array}$$

The construction is similar to the construction in the proof of Theorem 5. $\mathscr{T}_{\Sigma(M), \Pi, \mathscr{C}}$ becomes a $\Sigma', \Pi', \mathscr{C}'$ structure under the substitution $C \to C'$ and $\Sigma' = \Sigma[C/C']$, $\Pi' = \Pi[C/C']$ and $\zeta' = \zeta[C/C']$.

7.2. Operational Semantics

The operational model is naturally based on dynamic dispatch. In the standard object-oriented paradigm this means that selection of a method to execute a message is determined at run-time. The selection is typically based on the run-time type of the receiver object. In the paradigm of temporal classes this means that the relevant constraints are selected based on the run-time type of the receiver object (the object denoted by *self*). This is the single dispatch model.

The single dispatch model guarantees type safety if the conditions for subtyping are satisfied. In the case of embedding, the signatures of the inherited methods are not changed, and this is a particularly important case of subtyping. In this section we develop the semantic conditions for the preservation of behavior that apply to embedding.

Constrained matching does not guarantee type safety in the operational model based on single dispatch. As explained in Section 2.4, the selection of the relevant constraints is also based on the run-time type of the receiver object. But in addition, the run-time types of the actual parameters corresponding to the formal parameters

of *MyClass* are also inspected in order to make a type safe choice [7]. This is a limited form of multiple dispatch and it is provably type safe. In this section we prove under what conditions constrained matching guarantees dynamic preservation of behavior.

Suppose that an object of type T' has been substituted where an object of type T is expected. The clients expect behavior specified in T. Let $M' = (M_{S'}, \Sigma', \Pi', \mathscr{C}')$ be a model for T' and $M = (M_S, \Sigma, \Pi, \mathscr{C})$ be a model for T. Let ζ be a $\mathscr{C}$ constraint. If this constraint is evaluated for a substitution $\beta: X \to M$, where $M = UM'$, behavioral compatibility is not a problem. Note that $self \in X$. Indeed, in this case we have that *self* is assigned an object of type M, and the evaluation of the constraint ζ is performed in M.

Behavioral compatibility becomes an issue when the constraint ζ must be evaluated for a substitution $\beta': X \to M'$. In this case $self \in X$ will be assigned an object from M', and this applies to other variables as well. If $self \in M'$, then in either operational model $\phi(\zeta)[\phi(\beta')]$ will be evaluated in M'. $\zeta[\beta]$ denotes the constraint ζ in which the substitution β is carried out.

The above reasoning leads to the following conditions for the dynamic preservation of behavior for temporal classes.

PROPOSITION 11 (Dynamic conditions for perservation of bahavior).

Let $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be a theory morphism, and* $U: \mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ *its associated forgetful functor.*

Let M *be a model for* $(S, \leqslant, \Sigma, \Pi, \mathscr{C})$ *and* M' *a model for* $(S', \leqslant', \Sigma', \Pi', \mathscr{C}')$.

Suppose that we are given a substitution of variables $\beta': X \to M'$ *and let* $U(\beta') = \beta: X \to M$ *where* $M = UM'$.

The following conditions guarantee dynamic preservation of behavior, where ζ_{pre} *stands for a method precondition constraint and* ζ *stands for other type of constraint*:

1. $M \models \zeta_{pre}[\beta] \Rightarrow M' \models \phi(\zeta_{pre})[\phi(\beta')]$
2. $M' \models \phi(\zeta)[\phi(\beta')] \Rightarrow M \models \zeta[\beta]$.

Proof 14. 1. Before a method is executed, its precondition ζ_{pre} is evaluated. The expectation of the client is that this evaluation happens in M. But if $self \in M'$, the evaluation will actually happen in M'. In order to guarantee preservation of behavior as perceived by the clients, we must have $M \models \zeta_{pre}[\beta] \Rightarrow M' \models \phi(\zeta_{pre})[\phi(\beta')]$.

2. When method execution is completed, the outcome is described by the method postcondition constraint ζ_{post}. The clients expect that this constraint is evaluated in M. But under the above assumptions ($self \in M'$), what actually happens is that $\phi(\zeta_{post})$ is evaluated in M'. In order to guarantee preservation of behavior as perceived by the clients, we must have $M' \models \phi(\zeta_{post})[\phi(\beta')] \Rightarrow M \models \zeta_{post}[\beta]$. It is easy to see that this condition applies to all other types of constraints, i.e., class invariants, and history properties.

Note the contravariant nature of the requirement for method preconditions and the covariant nature of the requirement for the other types of constraints [35].

In the above conditions $\phi(\beta')$ denotes the result of transformation of the substitution of variables $\beta'\colon X \to M'$ under the morphism ϕ. For embedding and constrained matching this transformation assumes a particular form:

- Embedding: $X \subseteq X'$, $\phi(\beta') = \beta'$
- Matching: $X \setminus \{X_C\} \subseteq X'$, $\phi(\beta')\colon X[X_C/X_{C'}] \to M'$.

Consider now those cases in which the dynamic conditions for the preservation of behavior are provably satisfied. The first case is naturally the case of embedding of temporal theories. In this case we can prove that the dynamic conditions for the preservation of behavior are satisfied if the models for the two theories are related by the forgetful functor. So in this case static semantic conditions guarantee the dynamic conditions. This is exactly what one would like to accomplish.

PROPOSITION 12 (Dynamic conditions and embedding). *Let* $\phi\colon (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be an embedding morphism of temporal theories. Let* $M = (M_S, \Sigma, \Pi, \mathscr{C})$ *and* $M' = (M_{S'}, \Sigma', \Pi', \mathscr{C}')$ *be the corresponding models. If* $M = UM'$ *where* $U\colon \mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ *is the forgetful functor, the dynamic conditions for the preservation of behavior are satisfied.*

Proof 15. 1. Let $M \models \zeta_{pre}[\beta]$. We have $\phi(\zeta_{pre}) = \zeta_{pre}$ and $\beta' = \phi(\beta')$. We also have $M_s = M_{s'}$ for $s' \in S$ and thus $M_w = M_{w'}$. Hence $M' \models \zeta_{pre}[\beta']$, i.e., $M' \models \phi(\zeta_{pre})[\phi(\beta')]$.

2. Let $M' \models \zeta'[\beta']$. But $\zeta' = \zeta \sqcup \zeta'_{add}$. Since $M_s = M_{s'}$ for $s' \in S$ and $M_{w'} = M_w$, this implies $M \models \zeta[\beta]$.

The second case in which we can prove that static semantic conditions guarantee satisfaction of the dynamic conditions is the case in which two temporal theories are related by a morphism which is in fact constrained matching. Just as in the previous case, we can prove that the result holds if the underlying models of the two theories are related by the forgetful functor for constrained matching.

PROPOSITION 13 (Dynamic conditions and constrained matching). *Let* $\phi\colon (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *be a constrained matching morphism of temporal theories. Let* $M = (M_S, \Sigma, \Pi, \mathscr{C})$ *and* $M' = (M_{S'}, \Sigma', \Pi', \mathscr{C}')$ *be the corresponding models. If* $M = UM'$ *where* $U\colon \mathbf{Mod}_{\mathscr{C}'} \to \mathbf{Mod}_{\mathscr{C}}$ *is the forgetful functor, the dynamic conditions for the preservation of behavior are satisfied.*

Proof 16. 1. Let $M \models \zeta_{pre}[\beta]$. We have $\phi(\zeta_{pre}) = \zeta_{pre}[C/C']$. We have to prove $M' \models \zeta_{pre}[C/C'][\phi(\beta')]$. With $\beta'\colon X \to M'$ we have $\phi(\beta')\colon X[C/C'] \to M'$.

But $M_s = M_{s'}$ for $s' \in S \setminus \{C\}$ and $M_C = M_{C'}$, so that $M_{w'[C'/C]} = M_w$. This implies $M' \models \zeta_{pre}[C/C'][\phi(\beta')]$.

2. Let $M' \models \zeta'[\phi(\beta')]$. This means $M' \models \zeta[C/C'][\phi(\beta')] \sqcup \zeta'_{add}[\phi(\beta')]$.

Since $M_s = M_{s'}$ for $s' \in S \setminus \{C\}$ and $M_C = M_{C'}$, we have $M_w = M_{w'[C/C']}$ and thus $M \models \zeta[\beta]$.

The conditions in Propositions 12 and 13 are admittedly quite restrictive. But they do guarantee that static semantic conditions imply that the dynamic conditions for the preservation of behavior are satisfied.

The conditions for the preservation of behavior are often expressed in a weaker but more intuitive form [40]. If a class C' is derived from a class C by inheritance, behavioral compatibility requires that method preconditions in C' are weaker than the corresponding method preconditions in C. Other constraints (method post-conditions, class invariants and history properties) of C' are required to be stronger than the corresponding constraints in C. In order to compare the strength of two constraints, they must be evaluated in the same model. This evaluation necessarily happens in a model for the class C'.

The above reasoning is expressed by the following weaker conditions for the preservation of behavior.

DEFINITION 18 (Weaker dynamic conditions). Let $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ be a morphism of temporal theories.

Let $M = (M_S, \Sigma, \Pi, \mathscr{C})$ and $M' = (M_{S'}, \Sigma', \Pi', \mathscr{C}')$ be the corresponding models.

1. $M' \models \zeta' \Rightarrow M' \models \phi(\zeta)$
2. $M' \models \phi(\zeta_{pre}) \Rightarrow M' \models \zeta'_{pre}$

Note that the weaker conditions given above are easier to check syntactically. For example, these conditions could be enforced by a technique used in Eiffel [40] as follows:

- ζ' is $\phi(\zeta)$ extended by conjunction with additional constraints.
- ζ'_{pre} is $\phi(\zeta_{pre})$ extended by disjunction with additional constraints.

An alternative technique for dealing with the contravariant nature of the requirement for method preconditions is to require that they are the same in both theories, save for the changes made by a theory morphism. This is the limiting (minimal) case for satisfying contravariance. The situation is expressed formally in the next proposition.

PROPOSITION 14 (Invariant method preconditions). *If* $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *is a theory morphism such that* $\zeta'_{pre} = \phi(\zeta_{pre})$, *the weaker dynamic conditions are satisfied.*

Proof 17. Condition 1 is satisfied because ϕ is a theory morphism. Condition 2 is satisfied because $\zeta'_{pre} = \phi(\zeta_{pre})$.

The two corollaries that follow apply to embedding and constrained matching. All that is required in these two cases is that no additional method preconditions are introduced in $\mathscr{C}'$.

COROLLARY 5 (Embedding and method preconditions). *If* $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *is an embedding theory morphism and no additional method preconditions are allowed in* $\mathscr{C}'$, *the weaker dynamic conditions are satisfied.*

Proof 18. If ϕ is a theory morphism, Condition 1 is immediately satisfied. In addition, if ϕ is an embedding, we have $\phi(\zeta_{pre}) = \zeta_{pre}$.

COROLLARY 6 (Constrained matching and method preconditions). *If* $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *is a constrained matching theory morphism and no additional method preconditions are allowed in* $\mathscr{C}'$, *the weaker dynamic conditions are satisfied.*

Proof 19. If ϕ is a theory morphism, Condition 1 is immediately satisfied. In addition, if ϕ is constrained matching, we have $\phi(\zeta_{pre}) = \zeta_{pre}[C/C'] = \zeta'_{pre}$.

A result of a particularly pleasing form is obtained if in addition to the requirements of the above two corrolaries the underlying models of theories for embedding or constrained matching are related by the forgetful functor.

PROPOSITION 15 (Embedding and constrained matching). *If* $\phi: (S, \leqslant, \Sigma, \Pi, \mathscr{C}) \to (S', \leqslant', \Sigma', \Pi', \mathscr{C}')$ *is either an embedding or a constrained matching theory morphism such that no additional method preconditions are introduced in* $\mathscr{C}'$.

Let $M = (M_S, \Sigma, \Pi, \mathscr{C})$ *and* $M' = (M_{S'}, \Sigma', \Pi', \mathscr{C}')$ *be the corresponding models. Under these conditions the following properties holds*

1. $M' \models \zeta' \Rightarrow M \models \zeta$
2. $M \models \zeta_{pre} \Rightarrow M' \models \zeta'_{pre}$

where $M = UM'$, *with* U *the forgetful functor.*

Proof 20. If U is the forgetful functor such that $M = UM'$, then Condition 1 above is satisfied. In addition, for embedding and constrained matching we have $\zeta' = \phi(\zeta_{pre})$ where ϕ is a theory morphism. A particular construction of U guarantees that Condition 2 is satisfied as well.

We conclude the subsection on the operational model with comments on the actual implementation of *MyT*. *MyT* has been implemented on top of a persistent extension of the Java Virtual Machine. The implementation is described in detail in [8]. From the language viewpoint, there are two major differences between the presentation in this paper and the actual implementation as described in [8].

This paper does not address the issue of parametric polymorphism. *MyT* supports parametric polymorphism, even in its more sophisticated forms (bounded type quantification and F-bounded polymorphism). The technique for implementing parametric polymorphism in the Java technology is a major component of the actual implementation [8].

Multiple dispatch required for type-safe support of constrained matching has not been implemented in [8]. However, the fact is that the Java technology allows implementation of customized dispatch of this form. But the goal of the implementation [8] was to incorporate *MyT* into the Java programming environment as a nonconventional, declarative component. Temporal classes are compiled into standard Java class files. As such, they appear to the rest of the Java environment as Java classes. This is why single dispatch is used. Self-typing techniques from *MyT* are dropped because they are not compatible with the Java type system.

The run-time model for constraint evaluation is stack-oriented. It makes use of the Java stacks. A method invocation creates a new stack frame. A stack frame contains an operand stack which is used in evaluating constraints as explained below.

In a message $a.f(a_1, a_2, ..., a_n)$, $a, a_1, a_2, ..., a_n$ are terms which, when evaluated, produce object identifiers on the top of the operand stack. Before the selected method for f is invoked, $n+1$ object identifiers (the receiver and n arguments) are placed on the operand stack. Method lookup is based on the Java single dispatch mechanism. If f is a constructor, the result of method invocation will be the object identifier of the constructed object placed on the top of the operand stack after removal of $n+1$ top-level entries. If f is an observer, Boolean (in fact Java int) is placed on the top of the operand stack.

The implementation model enforces the temporal constraints. Before a method is invoked, the precondition constraints are enforced. After method execution, the postcondition constraints are enforced. Class invariants are enforced in all object states visible outside of a method execution. Initial constraints are used to properly initialize the state of a newly created object. Transition constraints and history properties are enforced for pairs of successive object states before and after method invocation. Rigid properties are also enforced. The details are necessarily the topic of a separate paper [8].

8. RELATED RESEARCH

A variety of other logic paradigms have been explored as a formal basis for an object-oriented language. This, in particular, applies to Horn clause logic with equality [2, 22] and other extensions of Horn clause logic (normal programs and programs [3]). F-logic [31] is possibly the most elaborate general logic framework addressing a variety of issues of the object-oriented paradigm, but it does not capture state changes.

The rewriting logic is the basis of the languages Maude [38] and MaudeLog [38]. Categorical aspects of behavioral specification in rewriting logic are presented in [15]. Dynamic logic has also been explored as a logic basis for the object-oriented paradigm [44].

The main advantage of a temporal paradigm is that it is event oriented, more expressive, and more natural in capturing simple and complex behavioral properties of objects. For example, temporal programming is certainly more intuitive than term rewriting. Unlike some other paradigms, temporal operators allow behavioral assertions on sequences of object states.

Temporal logic programming has been an area of active research [1]. First-order temporal paradigms have also been explored for the specification of program properties [32, 33]. To our knowledge, hardly any of that has been related to a typed object-oriented paradigm. Some recent results [17, 18] are at best object-based, rather than object-oriented. They do not capture inheritance or subtyping, which is where any model theory has major difficulties.

In comparison with our earlier work [5], the constraint language *MyT* presented in this paper is very different. Unlike the constraint language in [5], *MyT* is a truly object-oriented language, and so is its model theory. Its distinctive feature is that it integrates behavioral, temporal logic constraints, with results on object-oriented type systems. The operational model is also very different, as the operational model

for *MyT* is object-oriented, based on dynamic dispatch. These features were particularly relevant in making it possible to implement the language on top of the Java Virtual Machine [8].

In the model theory presented in this paper, the equality has been dropped since in the object-oriented paradigm equality can have different meanings from the usual interpretation. Although the temporal paradigm in [5] is the same, the form of the constraints in *MyT* as defined in Section 4.1 reflects the types of assertions suitable for an object-oriented programming language [40]. This allows results and techniques for the preservation of behavior in the inheritance hierarchies, such as those motivated by the notions of behavioral subtyping [35]. A major novelty in comparison with [5] is that this paper explores, for the first time, the view of classes as temporal theories. This, in particular, allows a model-theoretic approach to the issue of preservation of behavior in inheritance hierarchies.

In the model theory for *MyT*, constructors are represented as functions, and observers and mutators as predicates. This distinction has an interesting categorical interpretation following [28, 29]. An object type together with a collection of constructors may be viewed as an algebra of a suitably defined functor that captures constructor signatures. An object type together with a collection of observers and mutators may be viewed as a coalgebra of a functor representing signatures of observers and mutators. If $T: Set \to Set$ is a functor that captures constructor signatures, then its algebra is a function (in general a morphism) $T(X) \to X$ where X is the carrier of the underlying object type. If $T: Set \to Set$ is a functor that captures observer and mutator signatures, then its coalgebra is a function (in general a morphism) $X \to T(X)$ where X is the carrier of the underlying object type.

The canonical model for constructors is an initial algebra. The canonical model for mutators and observers is the terminal coalgebra. Although this is an interesting duality, the overall approach in [27–29] differs from the research presented in this paper in a number of essential aspects. In addition, the coalgebraic approach reported in [27–29] requires a Cartesian closed category equipped with coproducts. This technical requirement is not spelled out in the above cited work. The category of sets, of course, satisfies this requirement, but coming up with other suitable categories satisfying this requirement is not a simple matter at all.

In spite of the fact that constraints in [27–29] expressed in conditional equational logic, algebras and coalgebras in the above papers in fact do not capture these constraints. This is a major difference in comparison with the model theory presented in this paper.

Conditional equational logic used in [27–29] is inferior in its expressiveness in comparison with a temporal logic. The fact that it crucially depends upon the traditional notion of equality is in fact a problem for the object-oriented paradigm. Equality has several possible interpretations in the object-oriented paradigm. Imposing the requirement that every object type is equipped with traditional equality is much too strong and restrictive. Our approach assumes that if equality is available for an object type, it must be explicitly defined together with the constraints that specify its semantics.

The work reported in [27–29] is very limited with respect to the logic basis of the underlying paradigm. The category of specifications (theories in our paper) is

thus not used at all. Nonetheless, if T_1 and T_2 are specifications (theories) such that T_2 is derived by inheritance from T_1, at the level of models this is viewed as a forgetful functor G: $Models(T_2) \to Models(T_1)$. The fact that G has a right adjoint functor F: $Models(T_1) \to Models(T_2)$ is expressed by the cofree part of this adjunction. Given a model M_1 of T_1, a cofree model $F(M_1)$ of T_2 over M_1 is equipped with a canonical coercion ϵ_{M_1}: $GF(M_1) \to M_1$. Thus for any other model M_2 of T_2, and a coercion morphism f: $G(M_2) \to M_1$, there is a unique morphism h: $M_2 \to F(M_1)$ such that $G(h)\,\epsilon_{M_1} = f$, as in the diagram below.

$$\begin{array}{ccc} GF(M_1) & \xrightarrow{\epsilon_{M_1}} & M_1 \\ {\scriptstyle G(h)}\uparrow & & \uparrow = \\ G(M_2) & \xrightarrow{f} & M_1 \end{array}$$

The above pair of adjoint functors is not explicit in [29]. It actually reveals that from the algebraic viewpoint the coalgebraic approach presented in [28] is complementary to ours. But there is a significant difference. Our paradigm contains the logic, semantic part, which is missing from the (co)algebraic approach presented in [27–29].

An attempt to capture the temporal logic part is presented in [27]. The coalgebraic approach is extended with the temporal dimension based on the monoids of time. State changes are captured via actions of this monoid. Although interesting, this approach is limited in representing correctly the temporal dimension. State changes are not really related to the encapsulated procedures (mutators in our terminology). State changes are not related to the conditional equational constraints either. An integrated temporal logic-based algebraic paradigm is in our opinion significantly better.

Two obvious generalizations of the paradigm presented in this paper are multiple inheritance and parametric polymorphism. From the categorical viewpoint both are likely to be based on the pushout construction, already investigated in [21] and [26]. A parametric class may be viewed as a theory morphism $T \to T'$ where T stands for the type parameter and T' for the parametric class. A substitution of an actual class A for T is then another theory morphism $T \to A$. The result of substitution denoted $T'[A]$ is then the vertex of the pushout diagram:

$$\begin{array}{ccc} T & \longrightarrow & T' \\ \downarrow & & \downarrow \\ A & \longrightarrow & T'[A] \end{array}$$

This idea appeared first in [21]. The conditions on the order-sorted signatures that guarantee the existence of the above pushouts are investigated in [26], where the underlying logic is conditional equational. It appears that a temporal generalization should not be a problem, since [26] showed that the main problems for the existence of pushouts are in the properties of the underlying order-sorted signatures. Of course, this has to be carefully investigated.

The work based on pushouts is directly relevant to a model with multiple inheritance. Suppose that T' and T'' are classes (viewed as theories) that are both derived by inheritance from a class (theory) T. Then we have theory morphisms $T \to T'$ and $T \to T''$. If we now derive a class T''' by multiple inheritance from T' and T'', then we have theory morphisms $T' \to T'''$ and $T'' \to T'''$ rendering a pushout diagram.

The results on the preservation of behavior in the inheritance hierarchies of temporal classes are motivated in part by the notion of behavioral subtyping [35]. However, the results presented in this paper are quite different from the results in [35]. The first difference is that the results on the preservation of behavior for temporal classes are based on a formally defined notion of a temporal class as a theory. The second, closely related difference, is that the approach presented in this paper is model theoretic, i.e., it is based on algebraic models for temporal classes. This allows a much more rigorous treatment of the semantic, behavioral issues. And finally, the approach presented in this paper is not motivated by subtyping. Although embedding of theories is its particular case, constrained matching is a much more general and more flexible typing discipline, which is not subtyping at all. A substantial part of the results on the preservation of behavior applies to constrained matching.

The logic paradigm underlying the language TROLL [30] is considerably more general than the logic paradigm underlying *MyT*. TROLL is an object-oriented language for the specification of information systems. TROLL specification units (called templates) share a number of similarities with *MyT* temporal classes. TROLL templates include attributes (*MyT* has the observer predicates), events (corresponding to *MyT* mutators), and constraints (specifying the effects of events on attributes). Further behavioral features of TROLL include permissions (corresponding to method preconditions), obligations (specifying requirements at the end of object's lifetime) and patterns (process oriented specifications of permissible sequences of events). Invariant (static) constraints are expressed in the first-order predicate calculus. Dynamic constraints are expressed in a discrete, first-order temporal logic with infinite past. Furthermore, permissions allow specification of permitted sequences expressed in a temporal logic with finite past.

Developing an algebraic model theory of the sort presented in this paper, and an execution model, for the logic paradigm of this level of complexity, is a major challenge. In fact, our expectations are that the model theory developed in this paper extends to a temporal logic with finite past, but not to a full-fledged first order logic paradigm. A further distinction between TROLL and *MyT* is that a major goal of *MyT* is to integrate formal behavioral specifications with the results on object-oriented type systems. *MyT* supports the form of subtyping available in Java and C++, self-typing via a restricted from of matching, and even parametric, bounded, and F-bounded polymorphism. The latter are discussed in a separate paper [8]. Although TROLL does cover the issues of semantic inheritance, the model theory of *MyT* allows a much more careful and formal approach. These differences played no small role in implementing *MyT* on top of the Java Virtual Machine. Parametric polymorphic features of *MyT* prove to be essential in static typing of collections and queries [8].

At this point *MyT* is not a concurrent language, and its model theory is not intended to be either. Some, initial and informal considerations on concurrency in *MyT* have been investigated. A number of temporal paradigms for reactive and concurrent systems are available [37]. Research on synchronous languages [11] is also of interest. In fact, the examples of temporal classes in this paper are given in the style of synchronous paradigms. The relevance of results on temporal concurrent constraint programming [43] for the research presented in this paper remains to be investigated. As much as these paradigms are interesting and relevant, they are not object-oriented, and not tied to the results on type systems. The concurrent object-oriented paradigm raises subtle issues such as the inheritance anomaly [38]. This means that the preservation of behavior in inheritance hierarchies in concurrent object-oriented programming involves further subtleties. As the existing implementation of *MyT* is on top of the Java Virtual Machine, Java concurrent programming facilities provide an implementation support. For example, in order to be able to speak of the next object state in *MyT*, mutator methods should be implemented as Java synchronized methods.

9. CONCLUSIONS

In contradistinction to major difficulties in developing a model theory for full-fledged, typed procedural object-oriented languages, this paper shows that such a task becomes possible for a suitably defined declarative object-oriented language. Such declarative languages have an important place in typed object-oriented programming environments in prototyping, simulation and database applications [4, 6]. In addition, they have an appealing formal property of well-defined semantics.

The language, its applications, and the execution model have been covered in our related papers [6–8]. The rules of the type system follow the pattern presented in [7]. The model theory presented in this paper builds in part on our earlier paper [5]. Particularly interesting features of the model theory presented in this paper are standard categorical constructions used in its development. Indeed, the colimit model reflects precisely the temporal nature of the paradigm and represents a correct generalization of the classical initial model construction.

To our knowledge this is the first paper that presents and develops a view of classes as temporal theories. The same applies to our approach to inheritance, in which some inheritance relationships are proved to be morphisms of those theories. These developments explore some ideas presented in [21]. A more recent detailed analysis is given in [26]. Some informal ideas on specification morphisms also appeared in [16].

The view of classes as theories developed in this paper is a much more suitable paradigm for addressing the issues of preservation of behavior in the inheritance hierarchies. Furthermore, unlike other related approaches [35], the classes as theories view allows a model-theoretic approach lacking in other approaches to this problem [35].

A particularly important pragmatic development behind the theory presented in this paper is an implementation of *MyT* on top of a persistent extension of the Java Virtual Machine [8]. In order to integrate *MyT* into the Java programming

environment, self-typing was omitted because it is not compatible with the Java type system. This made it possible to compile *MyT* classes into Java class files. As such, compiled *MyT* classes appear to the rest of the Java programming environment as Java classes.

The two operational models for *MyT* presented in this paper are tied to the developed model theory. Furthermore, for these two models, the results on static semantics are related to the dynamic behavioral properties. Although just one of these two operational models has been implemented [8], it is in fact possible to implement both in the Java technology.

ACKNOWLEDGMENT

I thank Svetlana Kouznetsova for a number of useful comments which improved the paper. In particular, her observations led to a better proof of Lemma 1.

REFERENCES

1. Abadi, M., and Manna, Z. (1987). Temporal logic programming, *in* " Proceedings, Symposium on Logic Programming '87," pp. 4–16, IEEE Computer Society Press.
2. Alagić, S., Sunderraman, R., and Bagai, R. (1994). Declarative object-oriented programming: Inheritance, subtyping and prototyping, *in* "Proceedings, ECOOP '94," Lecture Notes in Computer Science, Vol. 821, pp. 236–259, Springer-Verlag, Berlin/New York.
3. Alagić, S., and Sunderraman, R. (1994). Expressibility of typed logic paradigms for object-oriented databases, *in* " Proceedings, BNCOD-12," Lecture Notes in Computer Science, Vol. 826, pp. 73–89, Springer-Verlag, Berlin/New York.
4. Alagić, S. (1995). A statically typed, temporal object-oriented database technology, *in* "Transactions on Information and Systems," Vol. 78, pp. 1469–1476, IEICE.
5. Alagić, S., and Alagić, M. (1997). Order-sorted model theory for temporal executable specifications, *Theoret. Comput. Sci.* **179**, 273–299.
6. Alagić, S. (1997). A temporal constraint system for object-oriented databases, *in* "Proceedings, Constraint Databases and Applications," Lecture Notes in Computer Science, Vol. 1191, pp. 208–218, Springer-Verlag, Berlin/New York.
7. Alagić, S. (1998). Constrained matching is type safe, *in* "Proceedings, the Sixth Int. Workshop on Database Programming Languages," Lecture Notes in Computer Science, Vol. 1369, pp. 78–96, Springer-Verlag, Berlin/New York.
8. Alagić, S., Solorzano, J., and Gitchell, D. (1998). Orthogonal to the Java imperative, *in* "Proceedings, ECOOP '98," Lecture Notes in Computer Science, Vol. 1445, pp. 212–233, Springer-Verlag, Berlin/New York.
9. Alagić, S. (1999). Temporal object-oriented programming, *Object-Oriented Systems* **6**, 1–42.
10. Baudinet, M. (1992). A simple proof of the completeness of temporal logic programming, *in* "Intensional Logics for Programming, Studies in Logic and Computation 1" (L. F. Del Cerro and M. Penttonen, Eds.), pp. 51–83, Clarendon Press, Oxford.
11. Berry, G., and Gonthier, G. (1992). The ESTEREL synchronous programming language: design, semantics, implementation, *Sci. Comput. Programming* **19**, 87–152.
12. Bruce, K. (1993). Safe type checking in a statically typed object-oriented programming language, *in* "Proceedings, Conference on Functional Programming," pp. 285–298, Assoc. Comput. Mach. Press, New York.

13. Bruce, K., Schuett, A., and van Gent, R. (1995). PolyTOIL: a type-safe polymorphic object-oriented language, *in* "Proceedings, ECOOP '95," Lecture Notes in Computer Science, Vol. 952, pp. 27–51, Springer-Verlag, Berlin/New York.
14. Cook, W. R., Hill, W. L., and Canning, P. S. (1990). Inheritance is not subtyping, *in* "Proceedings, Conference on Principles of Programming Languages," pp. 125–135, Assoc. Comput. Mach. Press, New York.
15. Diaconescu, R. (1996). Foundations of behavioral specification in rewriting logic, *Electron. Notes Theoret. Comput. Sci.* **4**, 225–243.
16. Ehrich, H. D., Denker, G., and Sernadas, A. (1993). Constructing systems as object communities, *in* "Proceedings, Theory and Practice of Software Development—TAPSOFT '93" (M. C. Gaudel and J. P. Jouannaud, Eds.), Lecture Notes in Computer Science, Vol. 668, pp. 453–467, Springer-Verlag, Berlin/New York.
17. Fiadeiro, J. L., and Maibaum, T. (1995). Sometimes "tomorrow" is "sometime": Action refinement in a temporal logic of objects, *in* " Proceedings, 2nd Int. Temporal Logic Conference '94," Lecture Notes in Artificial Intelligence, Vol. 827, pp. 48–66, Springer-Verlag, Berlin/New York.
18. Fisher, M. (1994). A survey of concurrent METATEM–the language and its applications, *in* "Proceedings, 2nd Int. Temporal Logic Conference," Lecture Notes in Artificial Intelligence, Vol. 827, pp. 480–505, Springer-Verlag, Berlin/New York.
19. Futatsugi, K., Goguen J., Jouannaud J., and Meseguer J. (1985). Principles of OBJ2, *in* "Proceedings, POPL '85" (B. K. Reid, Ed.), pp. 52–66, Assoc. Comput. Mach. Press, New York.
20. Goguen J., and Burstall R. (1992). Institutions: abstract model theory for specification and programming, *J. Assoc. Comput. Mach.* **39**, 92–146.
21. Goguen J. (1991). Types as theories, *in* "Topology and Category Theory in Computer Science" (G. M. Reed, A. W. Roscoe, and R. F. Wachter, Eds.), pp. 357–390, Clarendon Press, Oxford.
22. Goguen J., and Meseguer J. (1986). EQLOG: Equality, types, and generic modules for logic programming, *in* "Logic Programming: Functions, Relations and Equations" (D. Degroot and G. Lindstrom, Eds.), pp. 295–363, Prentice–Hall, Englewood Cliffs, NJ.
23. Goguen J., and Meseguer J. (1987). Unifying functional, object-oriented and relational programming with logical semantics, *in* "Research Directions in Object-Oriented Programming" (B. Shriver and P. Wegner, Eds.), pp. 417–477, MIT Press, Cambridge, MA.
24. Goguen J. and Meseguer J. (1987). Models and equality for logical programming, *in* "Proceedings, TAPSOFT '87" (H. Ehrig, G. Levi, R. Kowalski, and U. Montanari, Eds.), Lecture Notes in Computer Science, Vol. 250, 1–22, Springer-Verlag, Berlin/New York.
25. Goguen J., and Meseguer J. (1992). Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations, *Theoret. Comput. Sci.* **105**, 217–273.
26. Haxthausen A. E., and Nickl, F. (1996). Pushouts of order-sorted algebraic specifications, *in* "Algebraic Methodology and Software Technology (AMAST 96)" (M. Wirsing and M. Nivat, Eds.), Lecture Notes in Computer Science, Vol. 1101, pp. 132–147, Springer-Verlag, Berlin/New York.
27. Jacobs, B. (1996). Coalgebraic specifications and models of deterministic hybrid systems, *in* "Algebraic Methodology and Software Technology (AMAST 96)" (M. Wirsing and M. Nivat, Eds.), Lecture Notes in Computer Science, Vol. 1101, pp. 520–535, Springer-Verlag, Berlin/New York.
28. Jacobs, B. (1996). Inheritance and cofree constructions, *in* "Proceedings, ECOOP '96," (P. Cointe, Ed.), Lecture Notes in Computer Science, Vol. 1098, pp. 210–231, Springer-Verlag, Berlin/New York.
29. Jacobs, B. (1996). Objects and classes, co-algebraically, *in* "Object Orientation with Parallelism and Persistence" (B. Freitag, C. B. Jones, C. Lengauer, and H.-J. Schek, Eds.) pp. 83–103, Kluwer, Dordrecht.
30. Jungclaus, R., Saake, G., Hartmann, T., and Sernadas, C. (1996). TROLL—A language for object-oriented specification of information systems, *ACM Trans. Inform. Systems* **14**, 175–211.
31. Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundation of object-oriented and frame-based languages, *J. Assoc. Comput. Mach.* **42**, 741–843.

32. Kroger, F. (1987). "Temporal Logic of Programs," EATCS Monograph on Theoretical Computer Science, Springer-Verlag, Berlin/New York.
33. Lamport, L. (1983). Specifying concurrent program modules, *ACM Trans. Programming Languages Systems* **5**, 190–222.
34. Lloyd, J. W. (1987). "Foundations of Logic Programming," Springer-Verlag, Berlin/New York.
35. Liskov, B., and Wing, J. M. (1994). A behavioral notion of subtyping, *ACM Trans. Programming Languages Systems* **16**, 1811–1841.
36. Mac Lane, S. (1971). "Categories for the Working Mathematician," Springer-Verlag, Berlin/New York.
37. Manna, Z., and Pnueli, A. (1992). "The Temporal Logic of Reactive and Concurrent Systems," Springer-Verlag, Berlin/New York.
38. Meseguer, J. (1993). Solving the inheritance anomaly in concurrent object-oriented programming, *in* "Proceedings, ECOOP '93," Lecture Notes in Computer Science, Vol. 707, pp. 220–246, Springer-Verlag, Berlin/New York.
39. Meseguer, J., and Qian, X. (1993). A logical semantics for object-oriented databases, *in* "Proceedings of the ACM-SIGMOD Conference," pp. 89–98, Assoc. Comput. Mach. Press, New York.
40. Meyer, B. (1992). "Eiffel: The Language," Prentice–Hall, New York.
41. Orgun, M. A., and Wadge, W. W. (1992). Theory and practice of temporal logic programming, *in* "Intensional Logics for Programming, Studies in Logic and Computation 1," pp. 23–50, Clarendon Press, New York.
42. Plotkin, B. (1994). "Universal Algebra, Algebraic Logic and Databases," Kluwer Academic, Dordrecht/Norwell, MA.
43. Saraswat, V., Jagadeesan, R., and Gupta, V. (1996). Time default concurrent constraint programming, *J. Symbolic Comput*. **22**, 475–520.
44. Wieringa, R., de Jonge, W., and Spruit, P. (1994). Roles and dynamic subclasses: a modal logic approach, *in* "Proceedings, ECOOP '94," Lecture Notes in Computer Science, Vol. 821, pp. 33–59, Springer-Verlag, Berlin/New York.